Technical Report 1400
December 1990

Backplane Bus Distributed Realtime Clock Synchronization

D. R. Wilcox

NAVAL OCEAN SYSTEMS CENTER
San Diego, California 92152-5000

J. D. FONTANA, CAPT, USN
Commander

H. R. TALKINGTON, Acting
Technical Director

ADMINISTRATIVE INFORMATION

This work was performed by the Embedded Computer Systems Branch, Code 412, Naval Ocean Systems Center (NOSC), under Block Programs, Command and Control Department, Code 41.

Released by
L. J. Core, Head
Embedded Computer
Systems Branch

Under authority of
A. G. Justice, Head
Information Processing
and Displaying Division

ACKNOWLEDGMENTS

The work reported here benefited from the technical guidance and encouragement of many people. The author wishes to give special thanks to the following people:

Paul L. Borrill, National Semiconductor and chair of the Institute of Electrical and Electronic Engineers (IEEE) 896 Futurebus+ Committee, who communicated the author's concept of a "broadcast simultaneous sample command" to the IEEE P1212 Control Status Register (CSR) Committee where it was renamed the "strobe" concept and incorporated into their standard.

Hugh D. Copeland, NOSC, Code 552, who analyzed the feasibility of implementing the phase accumulation variable rate realtime clock model proposed for the IEEE P1212 CSR Standard as a single CMOS integrated circuit.

William P. Evertz, Synetics, who has spent innumerable hours reviewing, editing, and providing technical input for various IEEE and Navy realtime clock synchronization documents.

David V. James, Apple Computer and chair of the IEEE P1212 CSR Committee, who by encouraging the author to serve as editor to the realtime clock synchronization portion of the standard, made available technical input and feedback from members of the IEEE community.

Lui R. Sha, Advanced Real-Time Technology (ART) Project at the Software Engineering Institute of Carnegie Mellon University and chair of the IEEE 896 Futurebus+ Realtime Working Group, who acquainted the author with the development and advantages of rate-monotonic realtime scheduling technology.

Viet D. Tran, NOSC, Code 412, who fabricated and debugged an Am2942-based periodic phase modification variable rate realtime clock implementation designed by the author to demonstrate realtime clock synchronization across an IEEE 802.5 token ring local area network.

Richard A. Volz, Department of Computer Sciences at Texas A & M University, who edited the first draft documentation on realtime clock synchronization for the IEEE 896 Futurebus+ Committee and who contributed to that committee what is herein designated the hidden offset adjustable rate realtime clock implementation method.

JG

SUMMARY

This report presents methods of synchronizing the realtime clocks associated with a set of respective digital processing modules interconnected through a backplane bus. All the methods are based on the broadcast of a strobe signal over the backplane bus, which simultaneously samples all the realtime clocks being synchronized. The report describes and compares various methods of implementing realtime clocks, designed to permit their rate of advance with respect to physical time to be adjusted. It examines and makes recommendations on various interface standardization issues related to variable rate realtime clocks. It also gives and evaluates software algorithms for synchronizing variable rate realtime clocks.

CONTENTS

FIGURES

CONTENTS (continued)

FIGURES (continued)

1.0 INTRODUCTION

Many digital processing systems employ a sense of the physical time of day. Realtime systems use it to record the time at which events occur in the external world, to measure the elapsed time between events, to control the sequence in which event processing takes place, and to schedule initiation of events in the external world. Distributed fault-tolerant systems use it to detect events which fail to occur without incurring the cost of timeout messages.[1] Database systems use it to provide atomic coordination of transactions through the use of timestamps, as well as including it as an element of the data maintained within the database.

The software executing within a digital processor obtains a reading of the current physical time by accessing a realtime clock. A realtime clock is a register whose numeric content monotonically increases at a known rate with respect to physical time.[2] The simplest form of realtime clock consists of a processor-accessible counter, which increments on each cycle of an oscillator connected to it.

1.1 CENTRALIZED VERSUS LOCAL REALTIME CLOCK IMPLEMENTATION

An important type of digital processing system consists of two or more independent processors interconnected through a backplane bus. When more than one processor needs access to a realtime clock, the question arises as to where the realtime clock should be located. There are two basic approaches: providing a single centralized realtime clock and providing each respective processing module needing realtime clock access with its own local realtime clock. A combination of the two approaches is also possible.

It may seem better to employ a centralized realtime clock rather than replicating the realtime clock locally at each module. A centralized realtime clock requires less hardware overall. It also avoids the problem of synchronizing the collection of local realtime clocks. There are, however, three reasons why a distributed set of local realtime clocks may be a better approach.

First, access by a processor to a centralized realtime clock is generally slower than access to a local realtime clock connected more directly to the processor. The degradation in speed is due to the physically longer access path to the centralized realtime clock and the many potential software and hardware interfaces between the processor and the centralized realtime clock.

Second, time values obtainable from a centralized realtime clock are generally less accurate than those obtainable from a local realtime clock. The centralized realtime clock, by definition, is a shared resource. Contention may arise between users of the centralized realtime clock or on some component of the access path to the centralized realtime clock. Since competing traffic may be neither interruptible nor of known duration, it is often impossible to determine when the request for time data actually reached the centralized realtime clock. This calls into question the accuracy of the time data obtained.

---

[1] The basic idea is that message traffic through the distributed system can be reduced by having processes agree in advance on the physical time at which the failure of a process is assumed rather than sending timeout messages when failures are detected. See Lamport, L. April 1984. "Using Time Instead of Timeout for Fault-Tolerant Distributed Systems," *ACM Trans. Prog. Languages*, vol. 6, no. 2, pp. 254-280.

[2] In addition to the realtime clock, there also exists the concept of a virtual time clock. Given two events that happen at different times, both clocks can be used to determine which event happens first. The clocks differ in that a virtual time clock does not quantitatively indicate the duration of time between the events, but only their relative sequence in time. This limited capability is usually adequate for atomic transaction coordination time-stamping. Virtual time is beyond the scope of this paper. See Jefferson, D. July 1985. "Virtual Time," *ACM Trans. Prog. Languages*, vol. 7, no. 3, pp. 404-425.

An argument may be raised that systems requiring very accurate time data employ a centralized time standard, such as a cesium clock, which is considerably more accurate than a collection of local realtime clocks driven by relatively low-cost crystal oscillators. Regardless of the internal accuracy of the centralized realtime clock itself, the issue of contention for the clock, or for the access path to it, degrading the *obtainable* accuracy is still valid. Such systems can employ the external time standard for accuracy, yet still gain the benefits of distributed local realtime clocks, provided that they implement tight synchronization between the local realtime clocks and the external time standard.

Third, a centralized realtime clock is less fault tolerant than a collection of local realtime clocks. The centralized realtime clock represents a single point of failure. A distributed collection of local realtime clocks not only provides a backup source of time data should the primary source fail, but also provides a means of validating time data by comparing realtime clock values.

Distributed local realtime clocks avoid the three problems described above, but introduce a new problem, the need to keep the local realtime clocks synchronized with one another or with an external time standard. The remaining sections of this report assume the distributed local realtime clock approach.

1.2 ADJUSTING CLOCK VALUE VERSUS RATE

There are two methods for adjusting a clock to bring it into synchronization. They can be illustrated by analogy to adjusting a mechanical clock powered by a spring. The first method is to move the hands on the clock face. This is equivalent to modifying the *value* displayed by the clock. The second method is to adjust the tension on the spring powering the clock (usually by a screw adjustment on the back). This is equivalent to modifying the *rate* at which the time displayed by the clock advances. One can also use a combination of the two methods.[3]

Once a clock value has been initialized, adjusting the clock rate is usually superior to adjusting clock value for several reasons.

First, adjusting the clock rate avoids backward adjustments in time value. When the clock has been running too fast, it displays a time beyond the time desired. Adjusting the clock value to the proper time implies moving the time value backward for an instant. This destroys the utility of the clock in applications that depend on timestamps to establish the temporal sequence of events or transactions. Adjusting clock rate, on the other hand, never causes the clock value to move backward. When the clock has been running too fast, the rate is adjusted so that the value advances more slowly. The time value gradually converges toward the proper value. As the time value approaches the proper value, future rate adjustments converge on the proper rate to maintain the proper value.

Second, adjusting the clock rate avoids abrupt discontinuities in the otherwise steady advance of clock value. The problem with abrupt discontinuities can be illustrated by a simple example. Consider computing the average speed of an object moving in a straight line between two points. The average speed is the distance between the two points divided by the difference in the time values sampled when the

[3]There is at least one historical instance in which both techniques were used. Pope Gregory XIII was concerned that the Julian calendar, which had a leap year every 4 years, was not keeping pace with the Spring equinox needed to compute the date for Easter. In 325, the year of the Council of Nicea, the Spring equinox was on 21 March. By 1582, it had slipped to 11 March. The pope effectively "moved the hands" of the clock (the calendar) by declaring that the day after 4 October 1582 would be designated 15 October 1582. He effectively "adjusted the spring" of the clock by declaring that years divisible by 100 but not by 400 would no longer be leap years. See appendix A to the translation of Ptolemy's "The Almagest" in *Great Books of the Western World*. 1952. Encyclopedia Britannica, Chicago, vol. 16, p. 467.

object was at each point. If the clock value is adjusted dramatically just after the object leaves the first point but just before the object arrives at the second point, the computed speed could be dramatically in error. Even worse, a backward adjustment in time value could cause the computed speed to be negative. If the time values sampled at the two points were the same, which would happen if the elapsed time between the two points was canceled by a backward time adjustment, the speed computation would require division by zero. Adjusting clock rate, rather than clock value, still introduces the possibility of error. The error, however, is minimal since a small difference between the clock rate and the proper rate does not allow much error to accumulate during a short period of time.

Finally, adjusting the clock rate, when properly implemented, minimizes the need to continually readjust the clock in the future. When the clock is running too fast or too slow, any adjustment of the clock value is only temporary. Since the clock is running at the wrong rate, the clock value, although initially correct, drifts from the proper value as time progresses. The clock continually needs readjustment. Only by correcting the clock rate itself can this drift be minimized.

1.3 STROBE VERSUS FILTER SYNCHRONIZATION

This report concentrates on a realtime clock synchronization technique called the strobe technique. The strobe technique can be summarized as follows. Periodically a realtime clock synchronization algorithm is executed. It is executed frequently enough that no realtime clock drifts beyond a margin of synchronization error selected by the system designer. For each iteration of the algorithm, one of the modules simultaneously broadcasts to all modules, including itself, a command requesting that the hardware in each module immediately sample and locally store the value of its respective realtime clock. This command is called a strobe. The strobe effectively takes a snapshot of the state of all the realtime clocks at a given instant in time. The strobe does not broadcast realtime clock time value data. It only signals the moment when the data are to be locally collected. Once a module detects that the realtime clocks have been sampled, it executes an algorithm, called the adjustment algorithm, which adjusts the local realtime clock to bring it into synchronization. There are many variations of adjustment algorithms. All of them, however, adjust the local realtime clock based on the difference between its sampled value and the sampled values from one or more other realtime clocks.

Figure 1 shows a block diagram of the hardware components required by the strobe technique. The strobe detector and the strobe register are unique to each bus type. The adjustable realtime clock, the sample register, and the optional strobe watchdog timer can employ a common design. For parallel backplane busses, the strobe detector is an address comparator that identifies a strobe by recognizing the broadcast address dedicated to the strobe function. For serial busses, the strobe detector is a sequential state machine, which identifies a strobe by recognizing the broadcast destination address dedicated to the strobe function. When the strobe detector detects access to the dedicated address, it generates a signal to load the current value of the adjustable realtime clock atomically into the sample register. The same signal can also generate an interrupt to the processor to indicate the presence of a new sample enabling the adjustment algorithm to proceed. The strobe register and the optional strobe watchdog timer are explained later.

When realtime clock synchronization is required across an interface, it is often the case that a strobe on one side of the interface cannot be instantaneously presented by the interface to the other side. The problem is further complicated when the other side connects to a shared resource where variable delays due to contention are present, such as connections to a multiplexed peripheral or to another bus. The problem is solved by providing the interface with an internal timer (not shown in figure 1) that measures the elapsed time the strobe consumes crossing the interface. The elapsed time then is used to correct realtime clock sample values obtained across the interface.

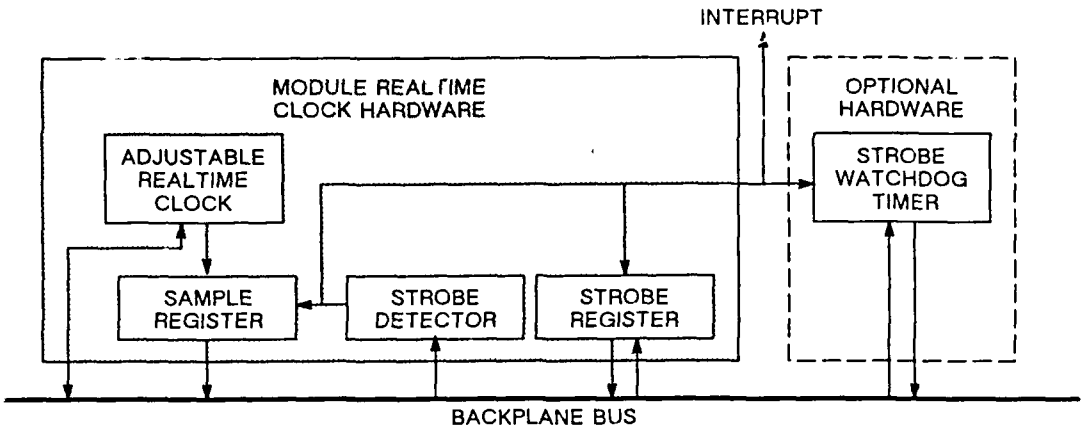


Figure 1. Strobe synchronization technique block diagram.

The synchronization of the realtime clocks on a set of interconnected modules requires that at least one module generate strobes. To avoid creating a potential single point of failure, at least some of the modules are equipped with both the ability to generate strobes and the ability to monitor whether strobes are, in fact, being generated. These modules have a watchdog timer.[4] The watchdog timer may be implemented either directly in hardware or as an application of the interval timing services provided by the operating system of the processor. A module initializes its watchdog timer when it goes online. A module reinitializes its watchdog timer whenever it detects the generation of a strobe by any module, including itself. A module generates a strobe when its watchdog timer interval expires. Normally, the module with the shortest watchdog timer interval generates the next strobe. This is because all watchdog timers are reinitialized at the same time by the same strobe and the watchdog timers on the other modules, whose initialized intervals are longer, do not have a chance to expire before being reinitialized. The approach provides fault tolerance since in the event that the module expected to generate the next strobe fails to do so, the module with the next successively longer watchdog timer interval automatically generates the next strobe. The approach also provides a means for the system to accommodate automatically the needs of the module whose realtime clock synchronization requires the shortest period between strobes.

The adjustment algorithm is intended to execute sometime during the interval between consecutive strobes. It is essential that each execution only compares realtime clock value samples obtained from the same strobe invocation. Mixing samples from different strobes, as explained below, destroys the simultaneous nature of the samples upon which the synchronization algorithm depends.

Samples from different strobes become intermixed when a new strobe appears while the adjustment algorithm is in the process of collecting samples from the previous strobe. This can happen in two ways. First, the adjustment algorithm execution may be delayed due to processor load. The delay can cause the

[4]The watchdog timer approach was presented in an earlier report as a means of providing fault tolerant strobe generation. See Wilcox, D. R. August 1989. "Periodic Phase Adjustment Distributed Clock Synchronization in the Hard Realtime Environment." Naval Ocean Systems Center, San Diego, TR 1310, p. 15. The original idea of watchdog timers with unique time intervals can be traced to the collision resolution method of the IEEE 802.3 Ethernet local area network. The use of strobe labels, presented below, eliminated both the need for the watchdog timer intervals to be unique and the need for the watchdog timers to be initialized all at the same time.

processor to miss the deadline required to complete collection of the sample values captured by the strobe before another strobe overwrites the sample values with new ones. Second, two or more strobes may be broadcast in rapid succession. This can happen for a number of reasons. Two or more watchdog timers may generate strobes at approximately the same time. A strobe may be delayed to such an extent that its broadcast appears at approximately the same time as the broadcast of another strobe. A strobe may appear from an interface to an external subsystem employing its own realtime clock synchronization. Regardless of the reason, when strobes appear in rapid succession the adjustment algorithm may be caught having only partially collected the sample values in response to one strobe before the another strobe overwrites the remainder with new sample values.

As an example, consider the interconnection of two independent backplane bus subsystems through an interface. Initially the interface between them is offline, isolating the subsystems from each other. Both subsystems independently generate their own strobes to synchronize their own realtime clocks. When the interface linking the subsystems first goes online, there is a critical initial period before a single source of strobe invocations has been selected when the modules of each subsystem potentially receive not only the strobe associated with their own subsystem but also the strobe originating from the other subsystem. The problem is even more severe in complex, loosely coupled networks consisting of many subsystems and interfaces.

Fortunately, there is a simple way to avoid these race conditions. When the strobe detector samples the realtime clock value in response to a strobe, it also atomically records information that uniquely identifies the strobe from other strobes. This unique information is called the strobe label. The content of the strobe label typically is derived from the source address of the module generating the strobe. The strobe detector records the strobe label in the strobe register. The adjustment algorithm checks to assure that all the realtime clock sample values required to be taken at the same instant of time have the same strobe label before performing computations using them. As shown in figure 2, if the strobe labels differ, the adjustment algorithm simply reaccesses the samples until they are all verified to be from the same strobe.

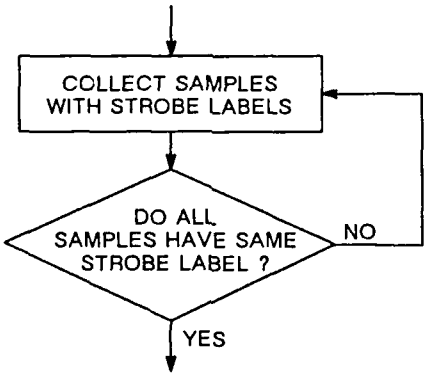


Figure 2. Strobe synchronization technique sample access.

The strobe technique relies on the presence of hardware to record immediately and simultaneously the values of the realtime clocks in the system upon detection of a strobe broadcast. If this hardware support is not available, another technique, called the filter technique, is generally used.[5] The filter technique employs statistics to estimate the relative difference between realtime clock values based upon a number of samples taken from each of the respective realtime clocks involved. In comparing the two techniques the strobe technique is superior in three respects.

First, the strobe technique avoids processor and backplane bus contention problems. Its adjustment algorithm needs to know how far ahead or behind one realtime clock is in relationship to another and, in the case of clock rate adjustment, how much time has elapsed between strobes. These parameters are determined by computing the numeric differences between realtime clock sample values. Once a realtime clock has been initialized, it is the differences between realtime clock sample values, not the realtime clock sample values themselves, which are critical to synchronization. As long as realtime clocks are sampled simultaneously and frequently enough to confine drift to within a specified bound, it is not important precisely when a strobe is broadcast. No harm is done when a strobe is delayed due to contention at the processor issuing the strobe or due to competing traffic on the backplane bus. The only area where contention is of concern is where *reception* of the strobe by the module causes the sampling of its local realtime clock value. Contention is avoided here through the use of dedicated hardware.

Contention is a major concern in the absence of hardware support. To some degree, the contention problem can be minimized by executing the software at a high priority. This is not a complete solution, however, because access to shared resources may not permit preemption regardless of the priority. Since the strobe technique avoids software and backplane bus contention problems, there is no need for the strobe to be broadcast at a high priority. The priority can be set to whatever level makes sense in relation to other processing. This flexibility makes the strobe technique compatible with rate monotonic scheduling technology.

Second, the strobe technique obtains realtime clock synchronization very quickly. When realtime clock synchronization error is plotted versus time, the *value* or the error is represented by a point on the curve and the *rate* at which the error changes is represented by the slope of the curve. This is depicted in figure 3. The slope of the curve is approximated by the slope of a line passing through two nearby points on the curve. The strobe technique can correct realtime clock value from the first strobe received and realtime clock rate from the first two strobes received.

The strobe technique synchronizes realtime clocks much more rapidly than the filter technique. Collecting all the realtime clock samples for the filter technique takes time, which makes the filter technique less robust, both at deriving the initial synchronization and at responding to sudden changes in realtime clock oscillator behavior. The resolution can be expressed in terms of a given standard deviation. The variance, which is the square of the standard deviation, is inversely proportional to the number of independent samples. Thus, the number of independent realtime clock samples required increases generally as the square of the relative realtime clock sample resolution to be statistically recovered. Furthermore, since the filtering technique is statistical, the confidence in the resulting synchronization is

[5] The filter technique uses statistics to improve the available realtime clock synchronization resolution. While the filter technique generally is used when there is no hardware support, it can also be used with hardware support to improve further the resolution provided through the hardware. For example, it can be used when the system needs a resolution greater than provided by some or all of the realtime clock oscillators.

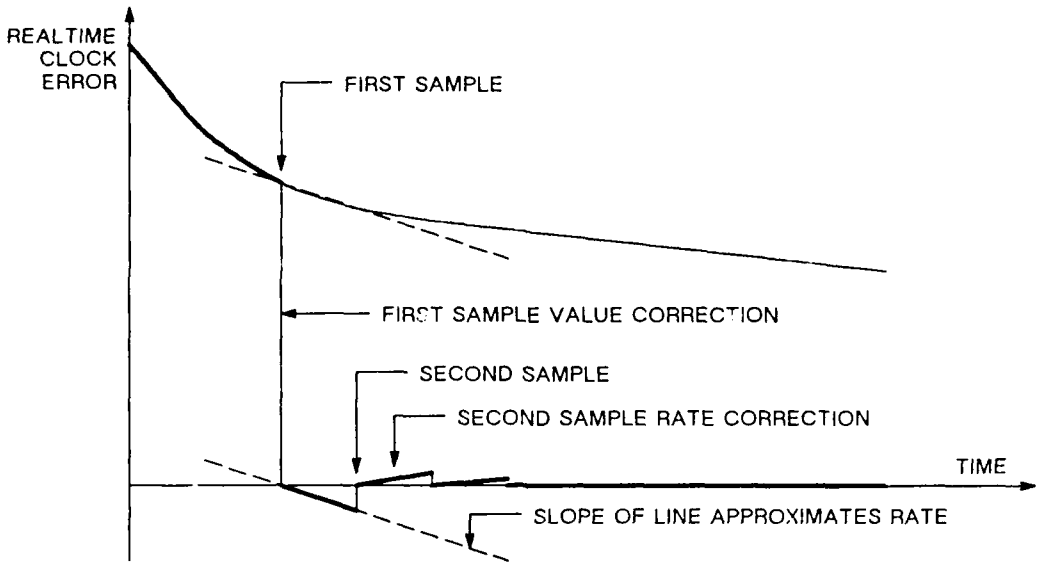


Figure 3. Strobe technique realtime clock error versus time.

also statistical. In theory, both techniques should be able to provide comparable resolution given enough time and a fairly stable oscillator.[6]

Finally, software support for the strobe technique is less complex than for the filter technique. This is because the strobe technique computes differences between realtime clock values directly rather than performing complex algorithms to estimate them statistically.

1.4 MASTER VERSUS AVERAGE REFERENCE TIME

The synchronization of two or more realtime clocks requires that their respective synchronization algorithms have a common understanding of what the correct time should be. Without this common understanding, attempts to synchronize the realtime clocks can be futile. To illustrate the point, consider two realtime clocks, X and Y, where X is initially running at a rate Δ *faster* than Y. The synchronization algorithm for X assumes that Y has the correct time. Since it has determined that X is running faster than Y, it *decreases* the rate of X by Δ to match the initial rate of Y. Independently, the synchronization algorithm for Y assumes that X has the correct time. Since it has also determined that X is running faster than Y, it *increases* the rate of Y by Δ to match the initial rate of X. Thus, two rate adjustments have been made, one at X and one at Y, both by the amount Δ. Rather than synchronizing the rates of X and Y, this process leaves X running at a rate Δ *slower* than Y. The problem is that the synchronization algorithms for

[6]The backplane bus environment is benign in comparison to that of a wide area network where the filtering technique must be used. "It should be recognized that clock synchronization requires by its nature long periods and multiple comparisons in order to maintain accurate timekeeping. While only a few measurements are usually adequate to reliably determine [*sic*] local time to within a second or so, periods of many hours and dozens of measurements are required to resolve oscillator skew and maintain local time to the order of a millisecond." Quoted from Mills, D. L. July 1990. *Network Time Protocol (Version 3). Specification, Implementation, and Analysis*. Univ. of Delaware, just above sec. 2.1, p. 5.

X and Y have different understandings of what the correct time should be. X assumes that Y has the correct time while Y assumes that X has the correct time. Even more troublesome, these definitions depend upon one another. This creates a realtime clock synchronization loop.[7] Synchronization of a given set of realtime clocks requires that there be only one definition of the correct time within the set so that all realtime clocks within the set are synchronizing to the same goal.

The synchronization algorithm's understanding of what the time should be is called its reference time. There are two important methods of determining reference time. The first method, called the master-slave method, assumes that one of the realtime clocks within a set of realtime clocks maintains the correct time. That realtime clock, called the master realtime clock, provides the reference time for the remaining realtime clocks, known as slave realtime clocks. The other method, called the averaging method, computes the reference time by averaging the time values of a set of realtime clocks. The components of the average may be weighted to accommodate some realtime clocks being judged better than others. The master-slave method can be viewed as a special case of the averaging method where the master realtime clock is given a weight of one and all the slave realtime clocks are given a weight of zero.

In the more general case, the synchronization of realtime clocks can be organized into a tree. Assuming the master-slave method, all realtime clocks except the one at the root of the tree are slave realtime clocks in relationship to their parent on the tree. All realtime clocks except the ones at the leaves of the tree are master realtime clocks in relationship to their children on the tree. This can be generalized to include the averaging method as well. Any realtime clock that has children on the tree can act as a master realtime clock for them. But their parent on the tree can also act as simply another component in the average computed by them as their common reference time. In such a structure, all the children of a given parent have a common reference time. That reference time is independent of the reference time used by the parent for its own synchronization. It is also independent of the reference times used by the children of other parents.[8] The tree organization avoids realtime clock synchronization loops.

Since the reference time must be unique, the independent synchronization algorithms associated with each of the realtime clocks must use the same realtime clock sample time values and compute the average in the same way to insure generation of the same result. The strobe technique is ideally suited for implementing the averaging method. The strobe broadcast samples all realtime clocks simultaneously. Each synchronization algorithm has access to the same collection of realtime clock sample values for a given strobe. The averaging method is dangerous for techniques that do not broadcast a strobe. Although adjustments can be made for the various times that samples are taken, the realtime clocks can continue to drift at unknown rates between the times these samples are taken making computation of a unique reference time impossible.

All else being equal, the averaging method generally gives a more accurate reference time than the master-slave method. As stated previously, accuracy is statistically improved by filtering. The averaging method implements a form of filtering not previously presented. It filters realtime clock sample values

---

[7] X is trying to synchronize using Y; and Y is trying to synchronize using X. This creates a realtime clock synchronization loop. The problem is similar to the deadlock of two tasks. Task A is waiting for a signal from task B; and task B is waiting for a signal from task A. This creates a coordination synchronization loop. In both cases, any kind of loop, not merely one with only two elements, creates a problem.

[8] The Network Time Protocol employs a tree organization. See Mills, D. L. July 1990. *Network Time Protocol (Version 3) Specification, Implementation, and Analysis*, Univ. of Delaware, sec. 2.2, pp. 7-8. The Network Time Protocol, following the convention of the telephone industry, calls the levels of the tree "stratum." Mills references Bell Communications Research, November 1986, *Digital Synchronization Network Plan*, Technical Advisory TA-NPL-000436. The most accurate realtime clocks, such as those controlled by cesium standards and radio clocks, are intended to be placed closest to the root.

obtained at the same time from various realtime clocks rather than realtime clock sample values obtained at various times from the same realtime clock. The reference time computed by the average is the filter output. In the master-slave method, on the other hand, the reference time is identical to the master realtime clock value. In the absence of filtering, the accuracy of each slave realtime clock value is no better than that provided by the master realtime clock to which it is synchronized.

After initialization, the presence of faulty realtime clocks can be detected by checking for realtime clock synchronization errors between realtime clocks that are beyond a range expected given the specification of the associated realtime clock oscillators. The master-slave method requires a mechanism to establish a new master realtime clock in the event that the current master realtime clock fails. The averaging method requires that the weights used by the averaging equation be adjusted proportionally to exclude failed realtime clocks.

2.0 ADJUSTABLE REALTIME CLOCK IMPLEMENTATION

All realtime clocks fundamentally consist of a counter, which is incremented by cycles of an oscillator. Processors generally use crystal oscillators to drive their realtime clocks. Crystal oscillator frequency is determined by factors such as manufacturing tolerance, temperature,[9] and aging.[10] From the circuit design point of view, crystal oscillator frequency is considered fixed at a value within some tolerance of the desired value. Since the oscillator is assumed to be nonadjustable, the only way to make the realtime clock adjustable is to adjust the way in which the counter counts. This section presents three alternative hardware approaches. They are the hidden offset method, the periodic phase modification method, and the phase accumulation method.

2.1 HIDDEN OFFSET METHOD

The hidden offset method[11] implements an adjustable realtime clock by correcting a least-significant portion of the realtime clock counter value not accessible by the realtime clock user. As shown in figure 4, the method partitions the n-bit realtime clock counter into an (n-m)-bit most-significant portion accessible by the user and an m-bit least-significant portion accessible only by the synchronization algorithm. The realtime clock value is restored to synchronization by adding to it the synchronization error. The realtime clock must be corrected frequently enough that its synchronization error never exceeds what can be represented by an m-bit two's complement number including sign bit. The addition only takes place when the m-bit least-significant portion of the realtime clock counter reaches a number consisting of a one followed by m-1 zeroes in the most-significant through the least-significant bit positions. As shown in figure 5, this number appears half way between the last time that the (n-m)-bit most-significant portion of the realtime clock counter was incremented (at time T) and the next time that it will be incremented (at time T+1). Although the realtime clock value may continue to drift while the synchronization error is held waiting for that number, the impact of the wait is negligible.

---

[9] For a discussion of quartz crystal temperature coefficient of frequency, see Radio Society of Great Britain. 1968. *The Radio Communication Handbook*, p. 6.2.

[10] Examination of crystal oscillator data sheets shows that aging is typically on the order of one part per million per year. Aging has little significance in comparison to manufacturing tolerance and temperature.

[11] The designation "hidden offset method" was assigned by the author. The method itself was obtained from an unpublished working draft paper, long since superseded, prepared by Richard Volz of the Univ. of Texas A&M for the IEEE 896 Futurebus+ committee. In that paper the method was called the "direct synchronization method." See Volz, R., D. Wilcox, and L. Sha. June 1989. "Maintaining Global Time in Futurebus+," *Guide to Real-Time Applications Using IEEE Futurebus+*, sec. 4.3.1, pp. 36-38.

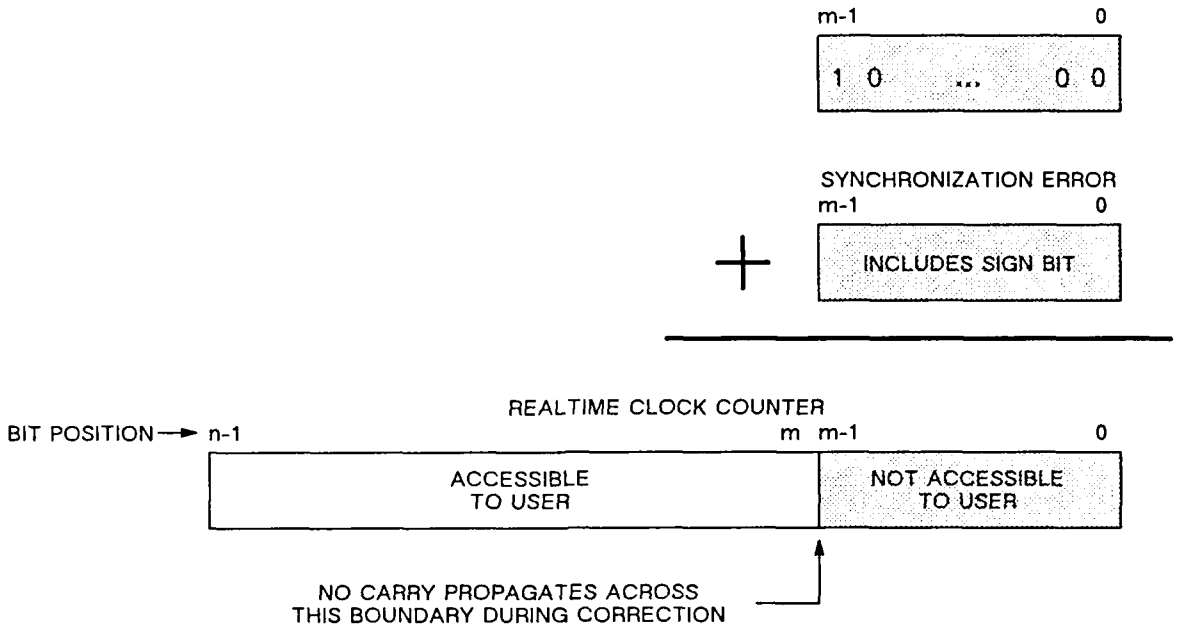


Figure 4. Hidden offset method variables.

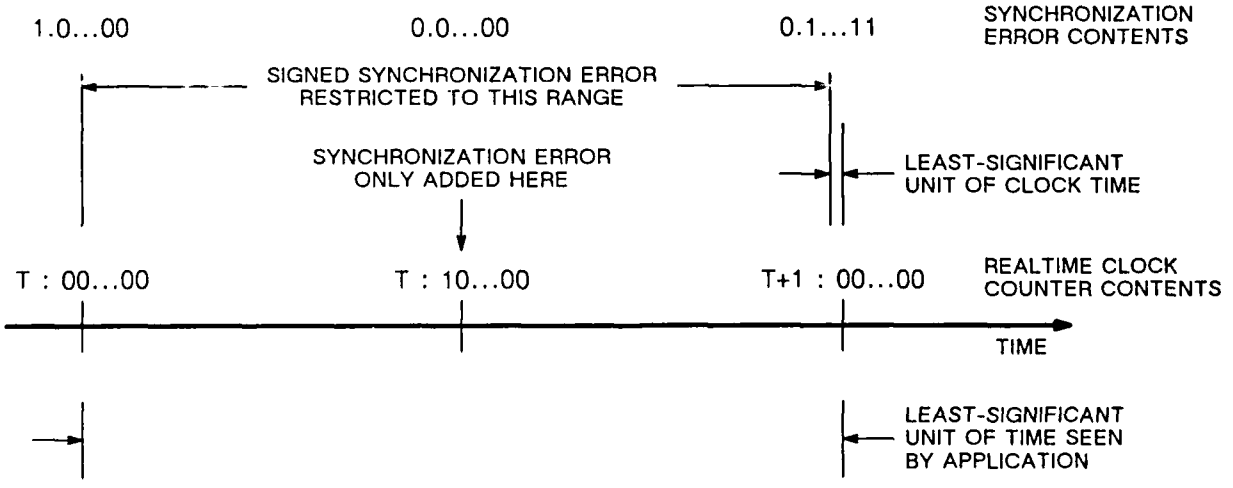


Figure 5. Hidden offset method correction.

As stated previously, the synchronization error is added to the realtime clock value to make the correction. Since the m-1 least-significant bits of the realtime clock value are required to be zero when the addition takes place, the addition simply replaces these bits with the bits from the corresponding bit positions of the synchronization error. And since the most-significant bit of the m-bit least-significant portion of the realtime clock value is required to be a one when the addition takes place, the addition replaces that bit with the complement of the synchronization error sign bit. There is no carry propagation from the m-bit least-significant portion of the realtime clock value into the (n-m)-bit most-significant portion of the realtime clock value. Thus, the addition reduces to loading new bits into the m-bit least-significant portion of the realtime clock counter.

Figure 6 shows a block diagram of an adjustable realtime clock implementation using the hidden offset method. When a correction is to be made, the adjustment algorithm loads the synchronization error into the synchronization error register. The load also sets a flip-flop to indicate that a correction request is pending. When both the detector recognizes the output 10...00 from the user nonaccessible m-bit least-significant portion of the realtime clock counter, and the flip-flop is in the correction pending state, a signal is generated to load the contents of the synchronization error register, with sign bit inverted, into the user nonaccessible portion of the realtime clock counter.

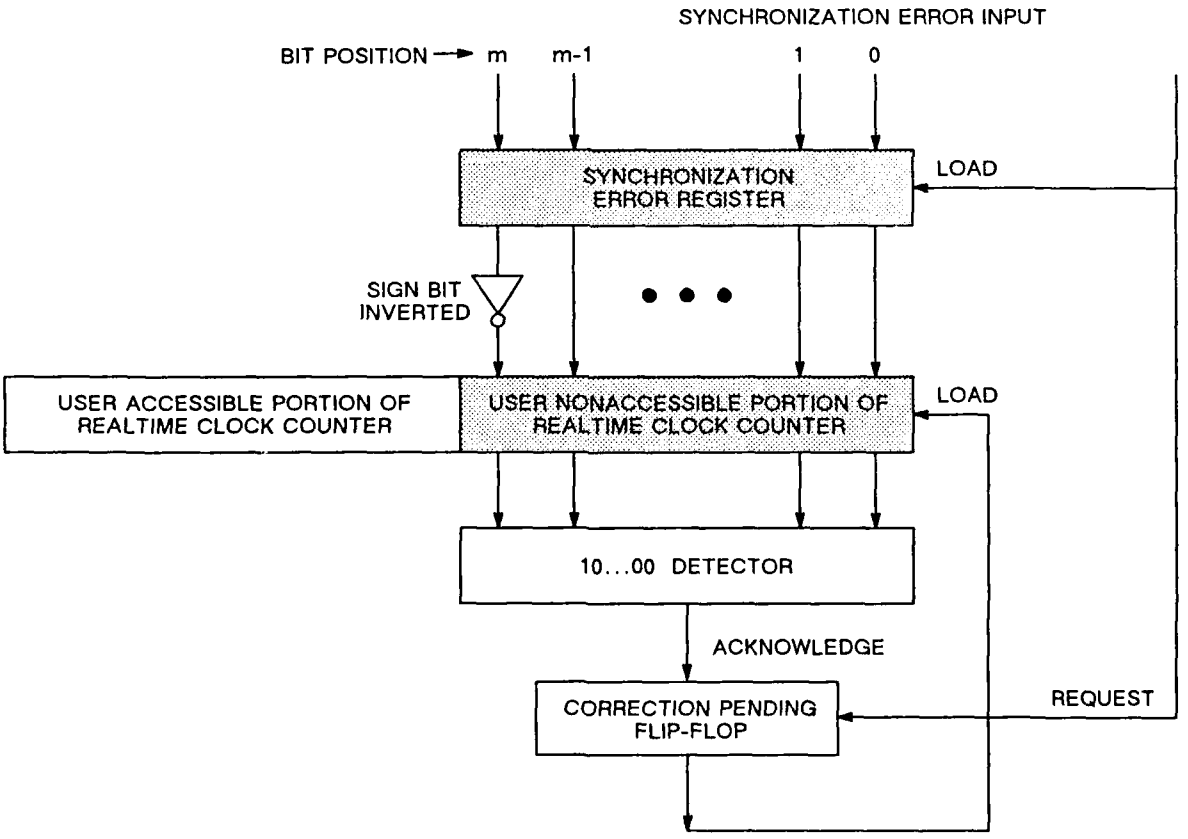


Figure 6. Hidden offset method block diagram.

The hidden offset method has the advantage of simple hardware implementation of its adjustable realtime clock. The method also has the advantage that its adjustable realtime clock is controlled directly by the synchronization error. Other methods, presented later, have adjustable realtime clocks controlled by parameters requiring complex computations, which are costly to implement in hardware. Since synchronization error is the difference between the local realtime clock value and the reference realtime clock value, it is easily computed in hardware. Thus, the hidden offset method offers the opportunity to implement the entire realtime clock synchronization process without the use of software. An entirely hardware implementation not only reduces software complexity and load, but eliminates the need for a processor on modules that otherwise have no need of one.

The hidden offset method has the disadvantage of restricting the realtime clock resolution accessible by the user. The method adjusts clock value rather than clock rate. Unlike other methods that adjust clock value, it avoids the problems of corrections causing clock values to go backward or to experience abrupt discontinuities. This is accomplished by making the user-accessible realtime clock resolution coarse enough that these problems are made invisible to the user. But, as with all methods that adjust clock value rather than clock rate, there still exists the need to continually make new corrections to compensate for drift due to the clock advancing at the wrong rate. Thus, the hidden offset method has the second disadvantage of requiring frequent corrections to maintain synchronization.

2.2 PERIODIC PHASE MODIFICATION METHOD

The periodic phase modification method[12] implements a variable rate realtime clock by inserting a programmable state machine, called a phase counter, between the realtime clock counter and the oscillator. Figure 7 shows a block diagram of this relationship. The phase counter continually cycles through a sequence of unique states. Each oscillator cycle causes the transition from one phase counter state to the next in the sequence or, in the case of the last state in the sequence, the transition back to the first state in the sequence. Whenever the phase counter enters the first state of the sequence, it generates an output signal, designated the increment enable, to increment the realtime clock counter. The phase counter acts like a frequency divider between the oscillator and the realtime clock counter. The realtime clock counter increments at a frequency equal to the frequency of the oscillator divided by the number of states in one cycle of the phase counter state sequence.

Figure 7. Periodic phase modification method phase counter.

The periodic phase modification method adjusts the realtime clock rate by periodically directing the phase counter to execute a cycle of its state sequence that contains either one more or one less than the

[12]In a previous report, this method was called the "periodic phase adjustment method." See Wilcox, D. R. August 1989. *Periodic Phase Adjustment Distributed Clock Synchronization in the Hard Realtime Environment*, TR 1310, Naval Ocean Systems Center, San Diego, pp. 4-8. The word "modification" has replaced "adjustment" because the latter tended to imply that once a phase cycle was adjusted, future phase cycles remained adjusted in the same way, which is not the case.

normal number of states. As shown in figure 8, inserting a state into the phase counter state sequence *lengthens* the period between the increment enable signals sent to the realtime clock counter for that cycle of the sequence. This *decreases* the rate at which the realtime clock counter increments. Similarly, deleting a state *shortens* the period between increment enable signals sent to the realtime clock counter, which *increases* the rate at which the realtime clock counter increments. Since most cycles of the phase counter state sequences have the normal number of phase states, this method permits small adjustments to be made in the *average* rate at which the realtime clock counter increments.

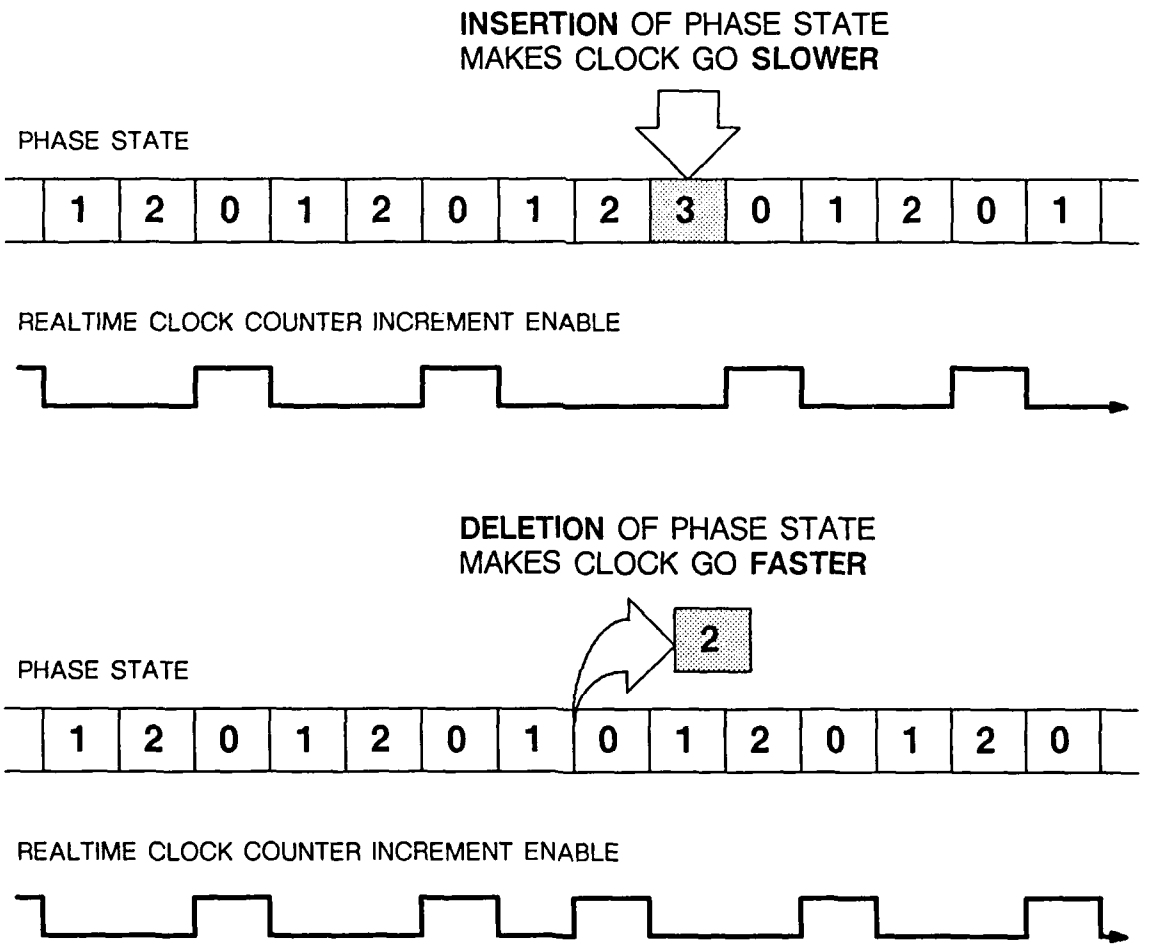


Figure 8. Modification of phase state sequence.

The realtime clock rate is adjusted by adjusting the frequency at which normal phase counter state sequence cycles are replaced with modified phase counter state sequence cycles. The period between modified cycle initiations is called the rate adjustment period. Figure 9 shows an example of how the inclusion of a modified cycle containing an additional phase state every rate adjustment period reduces the realtime clock rate. In this example, normal cycles have six phase states, modified cycles have seven phase states, and there are four cycles per rate adjustment period. These parameters will reduce the realtime clock rate by

$$1 - \frac{4 \times 6}{4 \times 6 + 1} = 0.04$$

or 4 percent of what it would have been without the introduction of modified cycles.

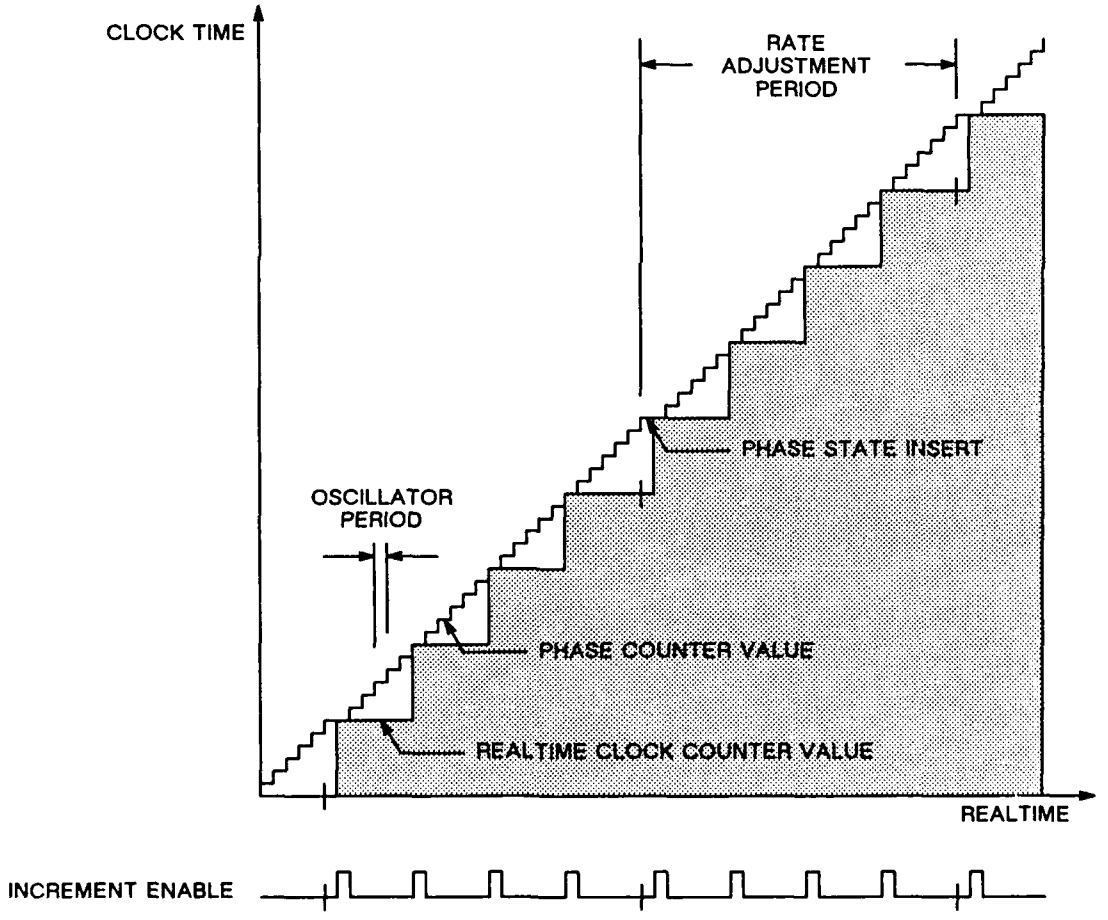


Figure 9. Rate adjustment period.

The periodic introduction of modified cycles can be facilitated in hardware by inclusion of a third counter called the rate adjustment counter. The rate adjustment counter indicates the number of normal cycles remaining before a modified cycle is introduced. The rate adjustment counter is decremented at the same rate as the realtime clock counter is incremented. When the rate adjustment counter reaches zero, the phase counter is directed to execute a modified cycle rather than a normal cycle. Associated with the rate adjustment counter is a rate adjustment reload register. The rate adjustment reload register contains the number of normal cycles per rate adjustment period. It provides the next value of the rate adjustment counter after it reaches zero. The adjustment algorithm software initializes the rate adjustment reload register. It also selects whether modified cycles are longer or shorter than normal cycles. The rate adjustment reload register provides a means of maintaining a constant realtime clock rate without further software intervention. Figure 10 shows a block diagram of a periodic phase modification adjustable rate realtime clock implementation, which includes the rate adjustment counter and the rate adjustment reload register.

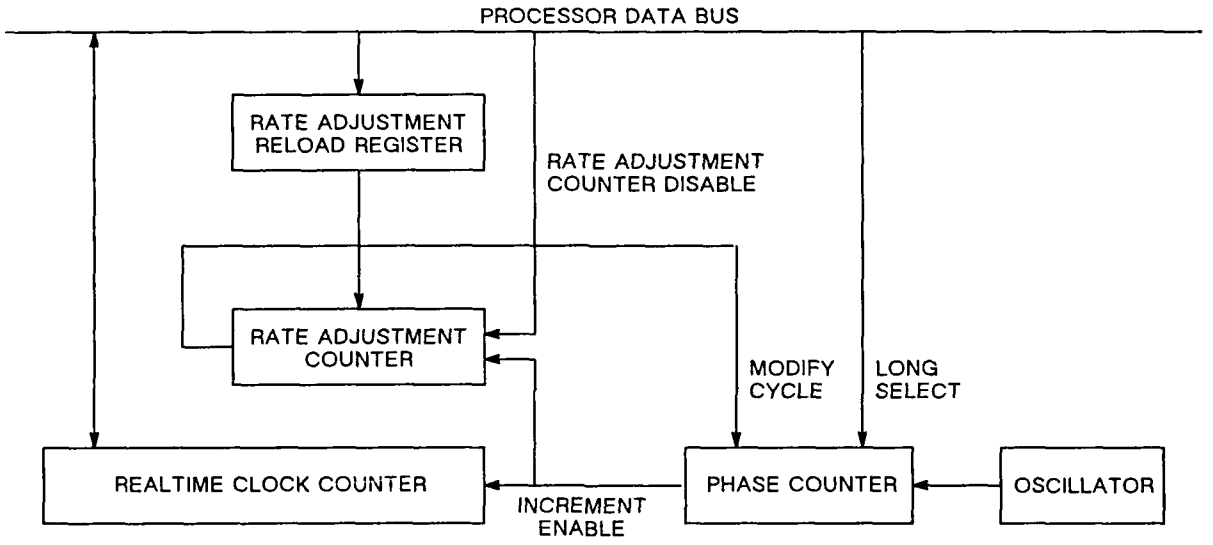
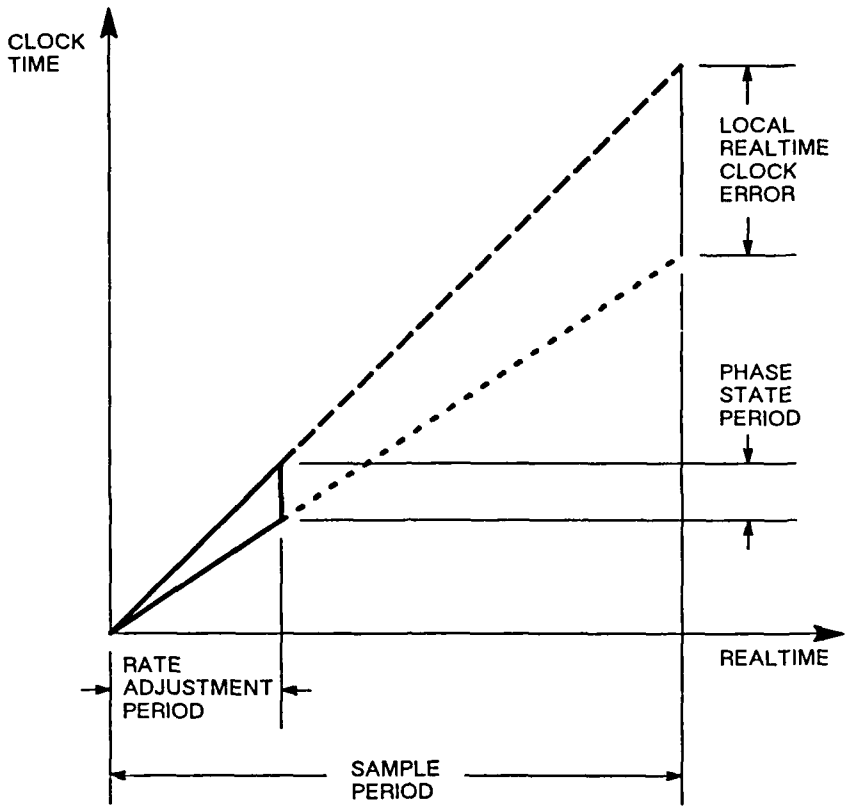


Figure 10. Periodic phase modification adjustable rate clock block diagram.

Figure 11 shows the computation of the rate adjustment period that keeps pace with the rate of the oscillator, assuming that there is initially no error in the realtime clock value. A slightly shorter rate adjustment period is needed to gradually decrease realtime clock synchronization error that may have already accumulated. The equation is a simple application of ratio and proportion or, when viewed graphically, of similar triangles.

Note that the local realtime clock error appears in the denominator of the equation. As the local realtime clock error decreases, the rate adjustment period increases. This is as one would expect, since as the realtime clock counter approaches perfect synchronization with the phase counter executing only normal cycles, the need for the phase counter to frequently introduce modified cycles decreases. If the realtime clock counter were perfectly synchronized with the phase counter only executing normal cycles,



$$\text{rate adjustment period} = \text{sample period} \times \frac{\text{phase state period}}{\text{local realtime clock error}}$$

Figure 11. Phase adjustment period computation.

the local realtime clock error would be zero and the rate adjustment period would be infinity. The equation fails at this point because it involves division by zero. Even when the local realtime clock error is not zero, but merely very small, the division can overflow the available quotient bits of an actual implementation. The problem is overcome by disabling the rate adjustment counter when the local realtime clock error falls below the threshold where overflow is known to occur. In the disabled mode, no modified cycles are introduced. The selection of the disabled mode is under the control of the rate adjustment algorithm software.

Integrated circuits performing a function similar to that of a realtime clock phase counter have been used for years in the design of general-purpose processors employing variable-length microcycles. The Am2925 integrated circuit[13] is a good example. The realtime clock implementation requires only one

[13]The Am2925 "Clock Generator and Microcycle Length Controller" was first introduced by Advanced Micro Devices, 901 Thompson Place, P.O. Box 3453, Sunnyvale, CA 94088.

output signal to increment the realtime clock counter. This is far simpler than the many phased output signals typically required by the circuitry of a general-purpose processor. Simple gates and flip-flops wired as a counter or shift register, rather than an integrated circuit specifically designed for variable-length microcycle processors, decrease cost and power consumption. Figure 12 shows an example of such a circuit, which implements the phase counter state sequences shown in figure 8. The upper and lower flip-flops in the figure implement the most-significant and least-significant bits, respectively, of the phase counter state.

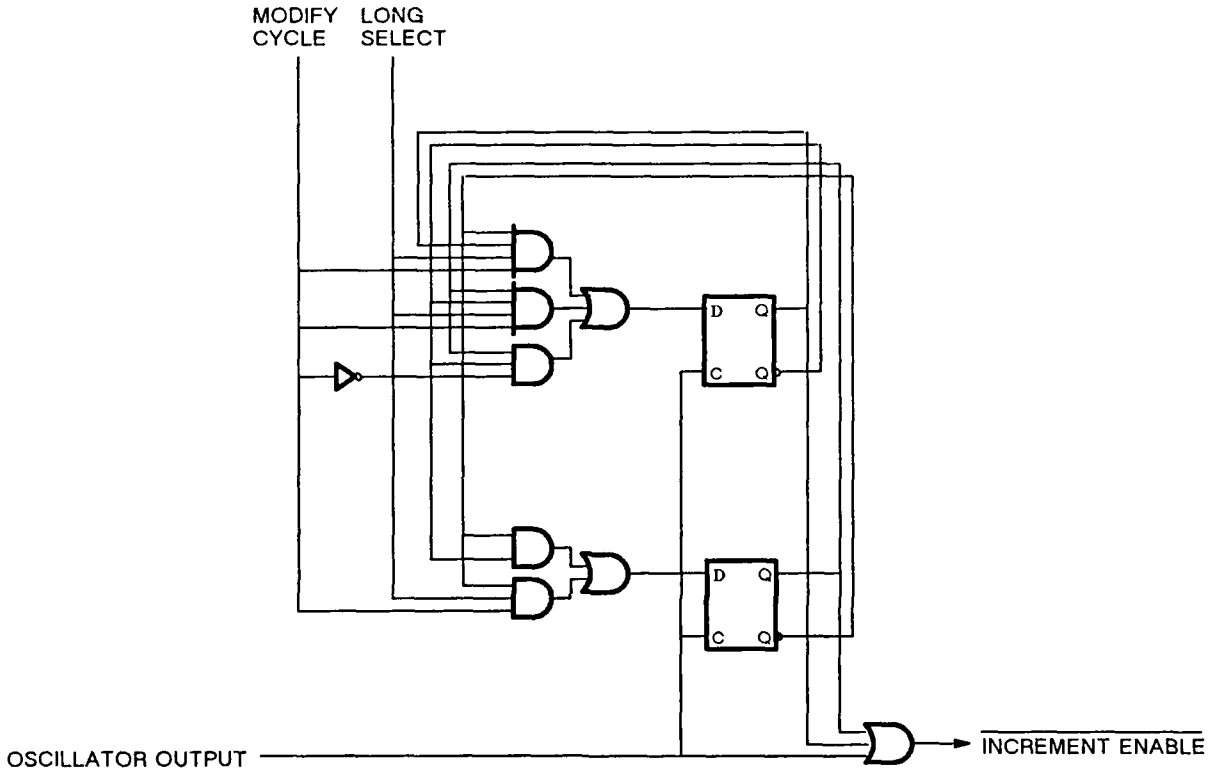


Figure 12. Divide-by-three phase counter implementation example.

Several Am2942 8-bit integrated circuits[14] can implement most of the remaining circuitry. A block diagram of the Am2942 is shown in figure 13. The Am2942 "address counter" and "word counter" are redefined to be the realtime clock counter and the rate adjustment counter, respectively. The Am2942 "done" signal is redefined to be the modify cycle signal, which in combination with a signal to select long or short cycles, controls the phase counter state sequence. The phase counter is external to the Am2942.

[14]The Am2942 "Programmable Timer/Counter DMA Address Generator" was first introduced by Advanced Micro Devices. The Am2940 is similar to the Am2942 but is not recommended because it has a physically larger package and does not permit control of the Word Count Register independently of the Address Register.

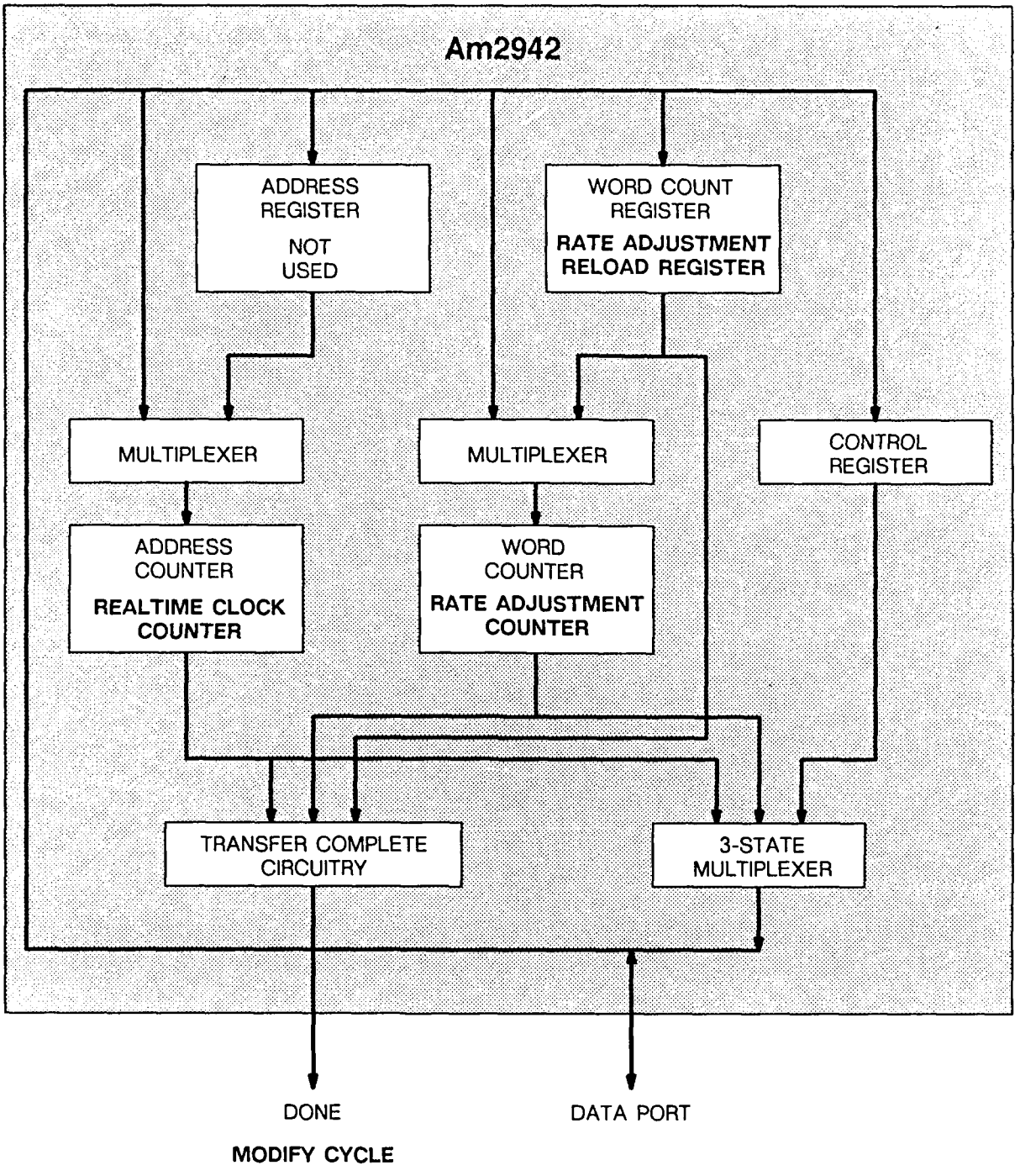


Figure 13. Am2942 integrated circuit block diagram.

The Am2942 "address register" is not used. The Am2942 "word count register" implements the rate adjustment reload register. The instruction set of the Am2942 supports the reload function. The sample register is external to the Am2942. Sample register access and processor access may be interleaved through the same Am2942 data port by assigning them different dedicated phases of the phase counter sequence.

2.3 PHASE ACCUMULATION METHOD

The phase accumulation method[15] implements a variable rate realtime clock by replacing the simple realtime clock counter with an accumulator and adder. The accumulator in combination with the adder performs a function similar to a counter, but rather than incrementing by one on each oscillator cycle, it increments by a programmable number called the tick period. Each cycle of the oscillator causes the contents of the tick period to be added to the contents of the accumulator and the sum to be stored back into the accumulator. At initialization, the adjustment algorithm sets the tick period to the nominal oscillator period. Once the adjustment algorithm has determined the realtime clock rate, it corrects it by adjusting the magnitude of the tick period. Figure 14 shows a block diagram of the phase accumulation method.

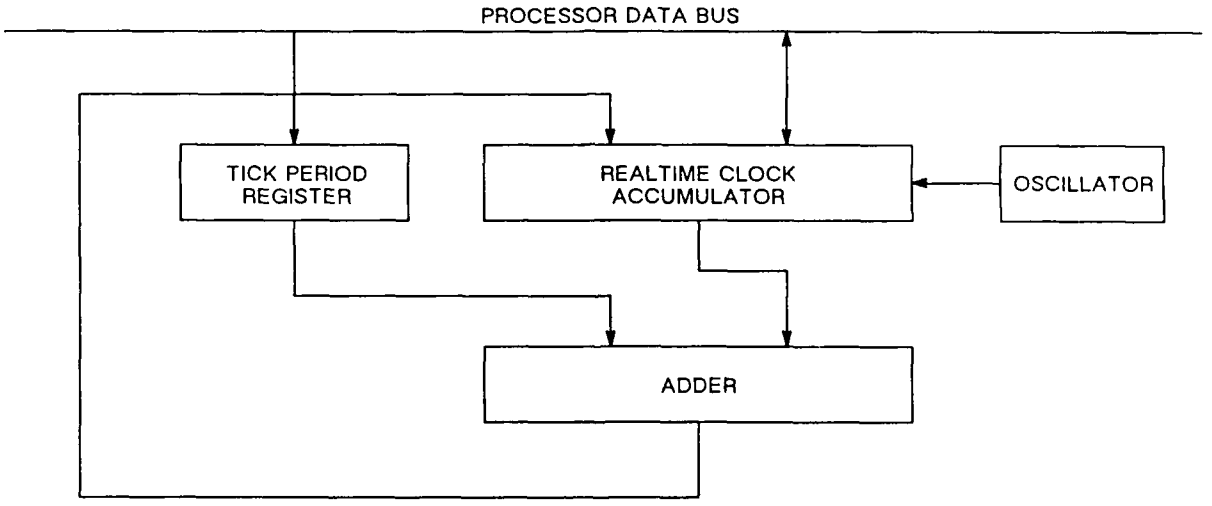


Figure 14. Phase accumulation adjustable rate clock block diagram.

The phase accumulation method has several advantages over the other methods. First, the phase accumulation method is conceptually simple. Comparison of the block diagrams of the respective methods illustrates this point. Second, the phase accumulation method permits the format of the realtime clock

[15]The basic concept of the phase accumulation method initially appeared in IEEE P1212, *CSR Architecture*, "Part I: an overview," Draft 0.31, September 18, 1989, sec. 11, pp. 65-66, and "Part II: a specification," Draft 0.30, September 5, 1989, sec. 6.1.12 - 6.1.16, pp. 34-37. These are unapproved IEEE drafts. Do not specify or claim conformance to these document.

value to be independent of the oscillator frequency. The periodic phase modification method requires the oscillator frequency to be an integer multiple of the frequency at which the realtime clock value visible to the user is incremented. The hidden offset method further requires that it be a power-of-two integer multiple. Given sufficient least-significant bits in the accumulator and adder to support a good approximation of the tick period, the phase accumulation method can accommodate any oscillator frequency. This flexibility often reduces hardware cost by permitting the same crystal oscillator to serve both the realtime clock and other module hardware, such as processors and interfaces, which usually have their own restrictions on oscillator frequency. Third, the phase accumulation method has the advantage of avoiding scaling problems within its adjustment algorithm, such as the divide overflow problem suffered by the periodic phase modification method described in section 2.2.

The primary disadvantage of the phase accumulation method is its requirement for a very wide adder and accumulator. A wide accumulator and adder are required to represent the tick period with sufficient accuracy to compensate for the accumulation of quantization error in the least-significant bits of the accumulator resulting from the potentially millions of serially executed additions between synchronization adjustments. Considerable error can accumulate in the least-significant bits over time.

The accumulator can be implemented using 8-bit register integrated circuits such as the SN74S374. The adder can be implemented using 4-bit arithmetic integrated circuits such as the SN74S381, in conjunction with look-ahead carry generator integrated circuits such as the SN74S182. Unfortunately, this approach leads to the use of a very large number of integrated circuits. A 96-bit-wide implementation requires 12 SN74S374s, 24 SN74S381s, and 8 SN74S182s for a total of 44 integrated circuits. Substitution of the 4-bit-wide SN74S281 for half of each SN74S374 and each full SN74S381, which combines the function of accumulator and adder within the same integrated circuit, reduces the total to 32 integrated circuits. The reduced integrated circuit count is offset by the physically larger package of the SN74S281.

Current integrated circuit technology easily supports implementation of the entire phase accumulation variable rate realtime clock within a single large scale integrated circuit. At the time of this writing, the author knows of no such implementation within the open market. A realtime clock consisting of a 96-bit accumulator, a 96-bit adder, a 64 bit tick period register, two 64-bit sample registers with independent strobe inputs, and a multiplexed 32-bit bidirectional data port has been partially designed to estimate the feasibility of implementation within a single integrated circuit.[16] Using the 1.2-micron National Security Agency (NSA) CMOS standard cell library, a military technology well within the capabilities of integrated circuit vendors, the design was estimated to fit within a square die 0.243 inch on a side. The maximum permissible oscillator frequency as seen by the accumulator and adder was estimated at 20 megahertz.[17]

2.4 SOFTWARE ADJUSTMENT METHOD

Both the periodic phase modification method and the phase accumulation method involve the design of variable rate realtime clock hardware. It is also possible to implement a variable rate realtime clock using software to correct the realtime clock value presented by nonadjustable realtime clock hardware. This method is called the software adjustment method. The software implementation is called a logical realtime clock to distinguish it from the hardware realtime clock.[18]

---

[16]This work was performed by Hugh D. Copeland, Naval Ocean Systems Center, Code 552.

[17]Faster 0.8 micro technology is widely available and even finer geometries are expected soon.

[18]The Network Time Protocol (NTP), which synchronizes realtime clocks within nodes interconnected through wide area or local area networks, encompasses a working example of the software adjustment method. See Mills, D. L. July 1990. *Network Time Protocol (Version 3) Specification, Implementation, and Analysis*, Univ. Delaware, Newark, sec. 5, pp. 39-45.

Associated with the local clock is a local clock service routine. The service routine is typically part of the operating system kernel. All requests for the current time pass through the service routine. The service routine converts the hardware realtime clock value into the adjusted realtime clock value returned to the user. The same conversion is also required for the realtime clock sample value captured in the sample register by the strobe.

The advantage of the software adjustment method is that it avoids the need to implement any special realtime clock hardware other than the sample register. The disadvantage of the software adjustment method is that all requests for either the current time or the sample time must pass through service routines. Service routine software access is less efficient than direct hardware access. It can also be functionally limiting. Controllers with little arithmetic processing capability of their own, such as direct memory access interfaces and input/output channels, can read the time directly from an adjustable realtime clock hardware memory-mapped interface. When this hardware is not present, these controllers must either also read the realtime clock adjustment parameters needed for later processing by an arithmetically superior processor or by some means interrupt such a processor to obtain the adjusted time results.

3.0 REALTIME CLOCK INTERFACE

This section presents some of the design issues related to backplane bus realtime clock interface standardization in general and to the IEEE P1212 Control Status Register (CSR) draft specification in particular.

3.1 REALTIME CLOCK ATOMIC ACCESS

The problem of atomic access to a realtime clock must be addressed when the access path to the realtime clock is narrower than the width of the realtime clock value. A common situation is 32-bit access to a 64-bit realtime clock value. Access to the entire realtime clock value requires a pair of accesses, one for the least-significant portion and the other for the most-significant portion. The problem is that a running realtime clock may propagate an internal carry from the least-significant portion to the most-significant portion between the time of the first access and the time of the second access of the pair. The internal carry propagation causes the realtime clock value constructed by concatenating the two accesses to be erroneous. The problem exists regardless of which portion is accessed first.

Several approaches designed to solve the problem, along with a summary of the advantages and disadvantages of each approach, are presented below. These are followed by a more detailed comparison of one approach against another.

In the **software read** approach, the software makes three accesses rather than two. First the most-significant portion of the realtime clock value is read, then the least-significant portion is read, and, finally, the most-significant portion is read again. It is assumed that all three reads take place in a time interval shorter than half the length of time between successive internal carry propagations from the least-significant portion to the most-significant portion of the realtime clock value. The software compares the first access with the third access. If they returned the same value, no internal carry propagation took place between the time of the first access and the third access. The realtime clock value is constructed by concatenating the most-significant portion read by either the first or the third access to the least-significant portion read by the second access. If, on the other hand, the first and third access obtain different values, an internal carry propagation did take place. If the software is willing to sacrifice some accuracy in the least-significant bits of the least-significant portion, it could simply perform a fourth access to read the least-significant portion again. It would then construct the realtime clock value by concatenating the values

read by the third and fourth access. A flow chart of this approach appears in figure 15. Alternatively, the most-significant bit of the value read by the second access is examined. If it is a *one*, carry propagation took place *after* the second access. The realtime clock value is constructed by concatenating the values read by the first and second access. If it is a *zero*, carry propagation took place *before* the second access. The realtime clock value is constructed by concatenating the values read by the third and second access. A flow chart of this alternative approach appears in figure 16.

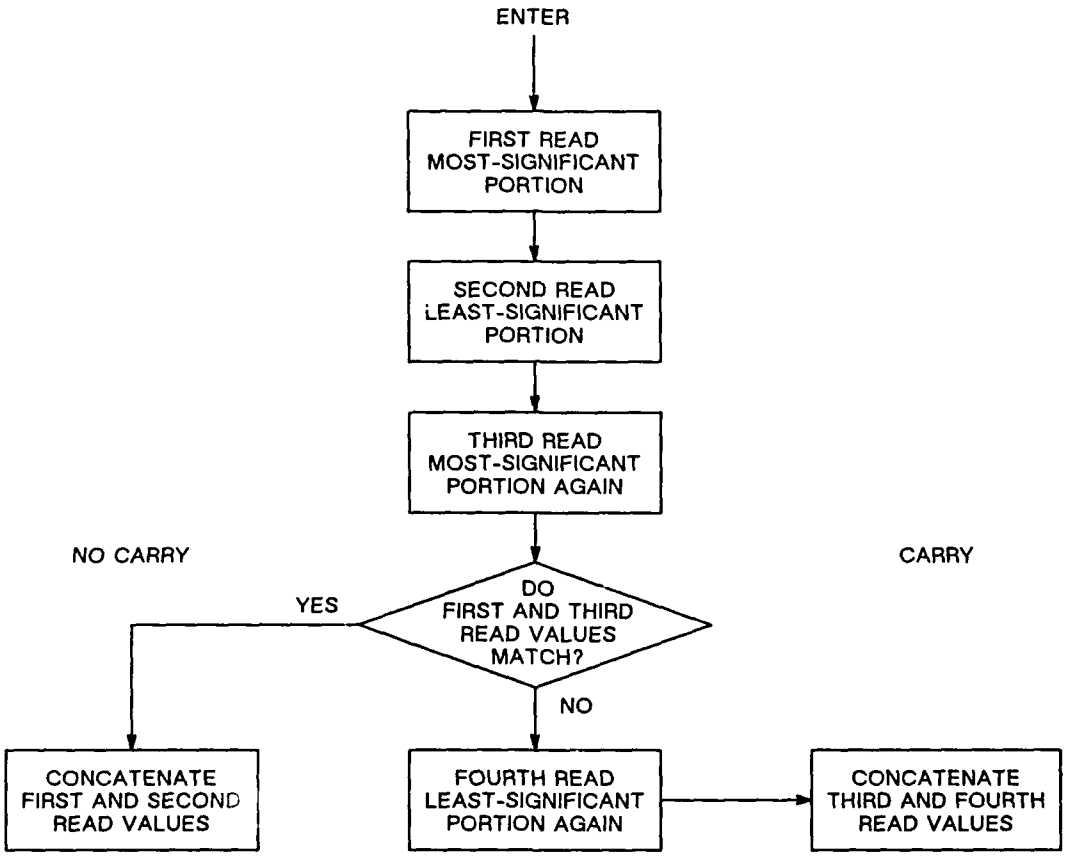


Figure 15. Software read realtime clock atomic access.

The advantages of the software read approach are (a) it minimizes the amount of hardware required, (b) it has no side effects, and (c) a requirement to read only from either the most-significant or the least-significant portion of the realtime clock value requires only one access. The disadvantages of the software read approach are (a) it requires three (or possibly even four) software accesses rather than two, (b) it requires software execution time, (c) it requires a processor to execute the software, and (d) it makes read access to the realtime clock dependent on the width of the access rather than being transparent.

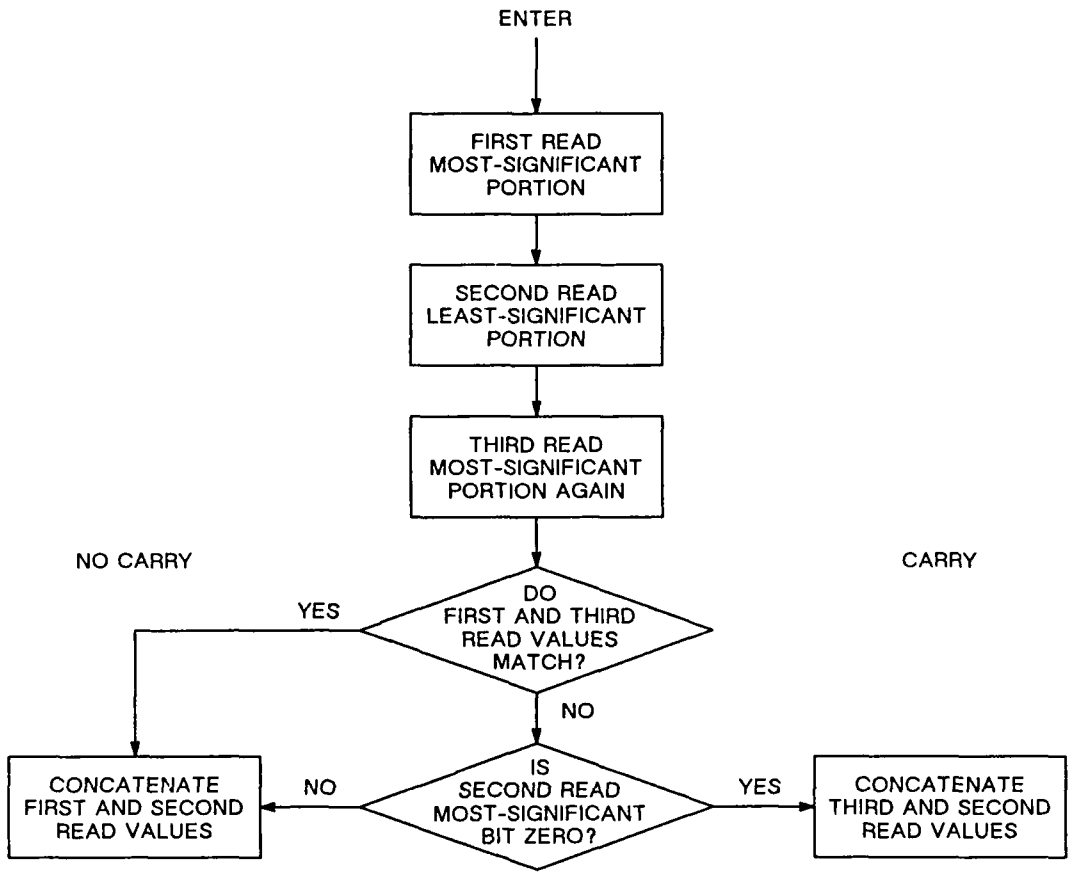


Figure 16. Alternative software read realtime clock atomic access.

In the **save register read** approach, a hardware register, called a save register, is inserted between the least-significant portion of the realtime clock value and the read access interface to it. The first software access returns the most-significant portion of the realtime clock value and, simultaneously, loads the least-significant portion of the realtime clock value into the save register. The second software access returns the least-significant portion of the realtime clock value saved in the save register by the first access. There is no internal carry propagation problem, since the entire realtime clock value is captured simultaneously by the first access. A block diagram of the save register read approach is shown in figure 17.

The advantages of the save register read approach are (a) it only requires two software accesses and (b) the realtime clock value can be read directly rather than requiring software processing. The disadvantages of the save register read approach are (a) it requires additional hardware for the save register and its control, (b) it introduces side effects (which are explained later), and (c) it is not possible to read from the least-significant portion of the realtime clock value without first reading from the most-significant portion of the realtime clock value due to the intervening save register.

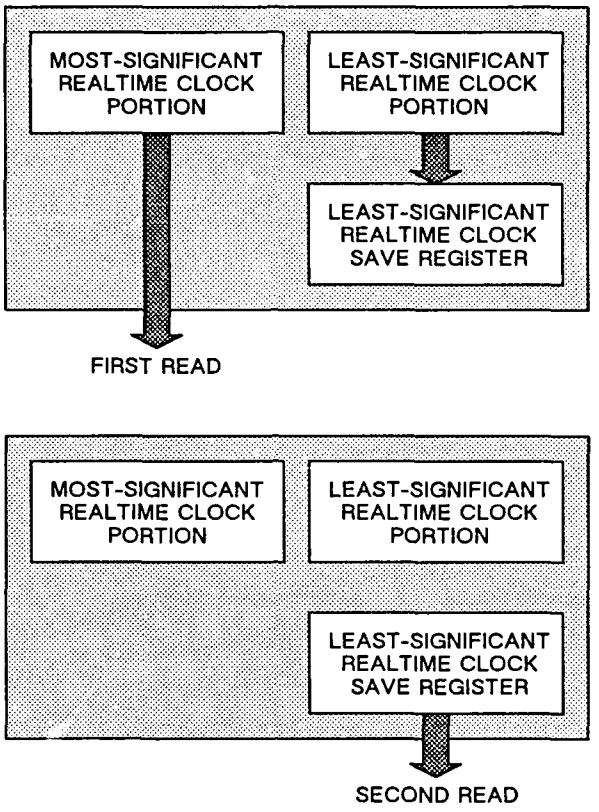


Figure 17. Save register read realtime clock atomic access.

In the **software write** approach, the software writes the least-significant portion before writing the most-significant portion of the realtime clock value. Since the software knows the value it is writing, it avoids writing a value into the least-significant portion that would be in danger of creating an internal carry propagation before the software has had a chance to write into the most-significant portion.

The advantages of the software write approach are (a) it minimizes the amount of hardware required, (b) it has no side effects, and (c) a requirement to write only into either the most-significant or the least-significant portion of the realtime clock value requires only one access. The disadvantages of the software write approach are (a) it restricts the flexibility of the value written, (b) it requires some software execution time, and (c) it requires a processor to execute the software.

In the **save register write** approach, a hardware register, called a save register, is inserted between the most-significant portion of the realtime clock value and the write access interface to it. The first software access only writes into the save register. The second software access writes into the least-significant portion of the realtime clock value and, simultaneously, transfers the content of the save register into the most-significant portion of the realtime clock value. There is no internal carry propagation problem since the entire realtime clock value is modified simultaneously by the second access. A block diagram of the save register write approach is shown in figure 18.

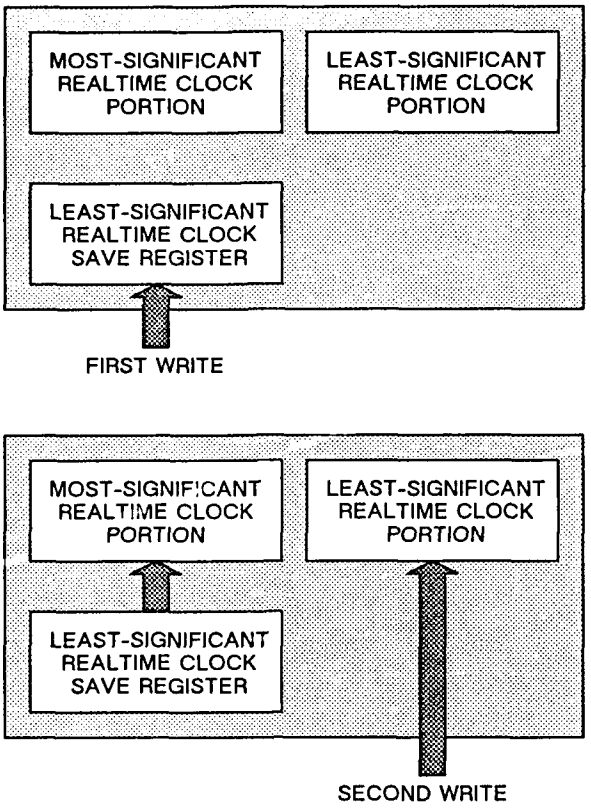


Figure 18. Save register write realtime clock atomic access.

The advantages of the save register write approach are (a) only two software accesses are required and (b) no special software processing is required. The disadvantages of the save register write approach are (a) it requires additional hardware for the save register and its control, (b) it introduces side effects, and (c) it is not possible to write to the most-significant portion of the realtime clock value without first writing to the least-significant portion of the realtime clock value.

In the **disable oscillator write** approach, the first software access writes the most-significant portion of the realtime clock value and, simultaneously, disables the oscillator. The second software access writes the least-significant portion of the realtime clock value and reenables the oscillator. Since the oscillator is disabled between the first and the second software accesses, there is no internal carry propagation to disrupt a partially written realtime clock value. The oscillator is not needed during this interval because whatever realtime clock value currently exists is being modified in any event.

The advantages of the disable oscillator write approach are (a) only two software accesses are required, (b) no special software processing is required, and (c) no hardware save register is required. The disadvantages of the disable oscillator write approach are (a) it requires more hardware than the software write approach and (b) it introduces a side effect.

Problems associated with the approaches summarized above are now explained and evaluated in more detail.

One of the disadvantages of the software read approach is the need for a processor to execute the software. Cost considerations may preclude including a processor within simple direct memory access

controllers or peripheral device interfaces simply to deal with the minimal computation associated with time-tagging their data transfers. When the software read approach is mandated, these devices can perform the accesses of the realtime clock interface associated with the software read approach without performing the actual computation of the time value. The processor expecting the data transfer then has the responsibility to execute the computation when it receives the data transfer. Unfortunately, this solution leads to another complication. A distinction is now necessary between realtime clock information obtained from devices returning the processed realtime clock value and those returning unprocessed realtime clock accesses.

The save register read approach, the save register write approach, and the disable oscillator write approach all have the disadvantage of introducing side effects. A side effect is an internal state that passes information between the first and second accesses of the approach. In the case of the save register approach, this information is passed through the save register. In the case of the disable oscillator write approach, this information is passed by the enabled or disabled state of the oscillator. While these approaches solve the carry propagation atomic access problem between the realtime clock and its backplane bus interface, their side effects create a new atomic access problem between their backplane bus interface and the processors that use it. Once a processor executes its first access to the realtime clock backplane bus interface, all other processors must be prevented from accessing that interface until after the processor executes the second access required by the approach to that interface. This is usually accomplished through a backplane bus locking mechanism.

The side effect problem can also be solved by providing an additional access path to circumvent the storage component responsible for the side effect when it is not needed. Figure 19 shows the additional path for the save register read approach. The storage component in this case is the save register. As previously described, atomic access to the full realtime clock value requires two accesses and to the most-significant portion of the realtime clock value requires one access. Access to the least-significant portion of the realtime clock value can be obtained with only one access by using the additional access path to circumvent the save register. A similar modification can be applied to the save register write approach. In the case of the oscillator disable write approach, the alternative path performs the same access as the path that disabled the oscillator without disabling it.

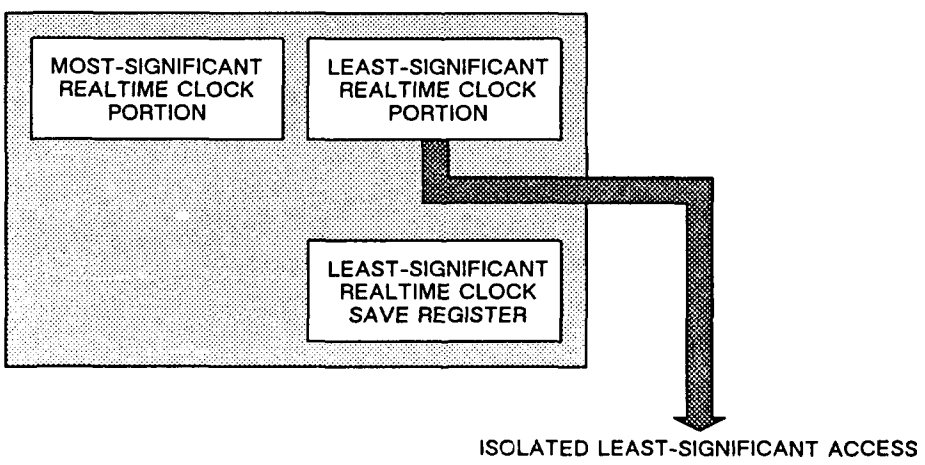


Figure 19. Modified save register read realtime clock atomic access.

The Institute of Electrical and Electronic Engineers (IEEE) P1212 Control Status Register (CSR) draft standard uses the phase accumulation method as its realtime clock model.[19] Figure 20 shows a block diagram of the model. The accumulator and adder are 96 bits wide. The user has access only to the 64 most-significant bits of the accumulator through the realtime clock value registers. These registers assume the software read and software write approaches when the access path to them is 32 bits wide. This decision was based on the desire to eliminate side effects and to minimize hardware implementation costs. The realtime clock model uses the save register write approach for access to its 64-bit tick period. The tick period specifies the amount that the realtime clock value increases on each cycle of the oscillator. The save register write approach was selected here because oscillator cycles occur so rapidly in comparison to software execution speed that the software write approach is not feasible.

3.2 TIME VALUE FORMAT

It is advantageous for all absolute time values within a system to be represented by the same format. This eliminates the overhead in memory space and execution time associated with the many permutations converting from one format to another. While no single format is going to please everyone, the advantages of a single format outweigh the consequences when viewed at the system level.

Many module designs employ the same crystal oscillator for both their arithmetic processor and their realtime clock. This saves the cost of an additional crystal oscillator and simplifies the circuitry that synchronizes processor access to the realtime clock. This suggests that the system time value format should be related to processor crystal oscillator frequency. Unfortunately, crystal oscillator frequencies vary among processors and among versions of the same processor. The least common denominator among most of them is 1 megahertz, or a period of 1 microsecond. Recall, however, that the rate of the realtime clock needs to be adjustable to permit synchronization with other realtime clocks in the system. Section 2.0 presented various methods of implementing variable rate realtime clocks. The whole purpose of these methods was to make the realtime clock rate independent of the frequency of the oscillator driving the realtime clock. The system time value format can be independent of processor oscillator frequency because the variable rate realtime clock is already independent of oscillator frequency. The advantages of using a single crystal oscillator can still be maintained.

While the hardware instruction sets of some processors are designed to handle decimal arithmetic, nearly all processors are designed to handle binary arithmetic. Therefore, a binary time value format is more universal and consequently a better standard.

The least-significant bit of the binary time value format could represent a common decimal fractional unit of time such as 1 millisecond, 1 microsecond, or 1 nanosecond. This would simplify application software involving the particular unit selected. But these possible selections are related to one another by the powers of 10 rather than the powers of 2. There is no convenient binary mapping between them. Thus, the software simplification is not general, but is limited to application of the unit selected.

The selection of a single time value format does not necessarily imply the selection of a single resolution for all clocks using that format. Resolution is the extent to which the *physical time* represented by two time values can be differentiated. A realtime clock value updated by an oscillator once per second

---

[19]See IEEE P1212 Draft 4.0, September 22, 1990, sec. A-7.3, p. A44. This is an unapproved draft. Do not specify or claim conformance to this document.

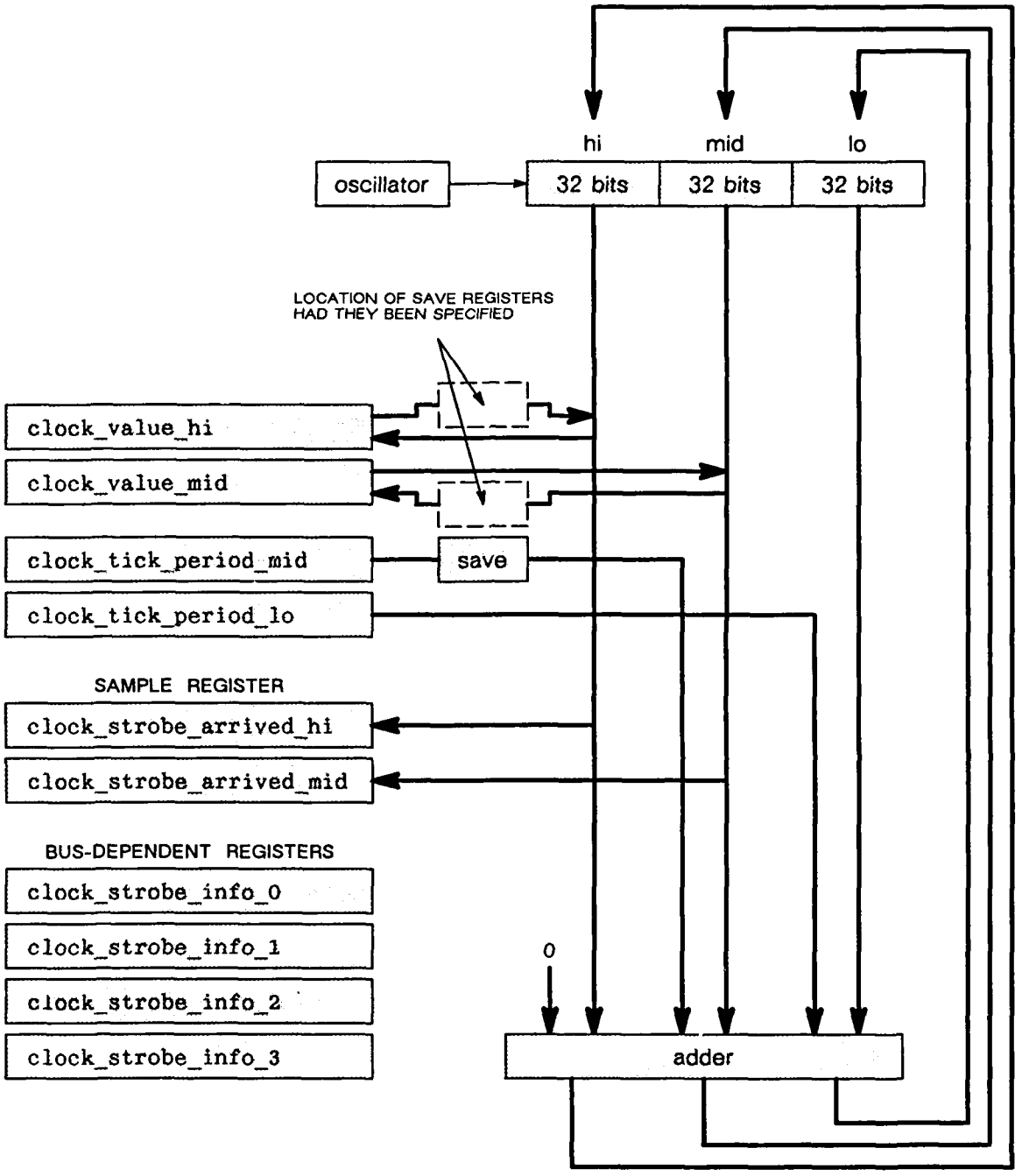


Figure 20. A proposed standard interface realtime clock model.

has a resolution of 1 second. No matter how many least-significant bits representing fractional parts of a second are included in the time value format, these additional bits convey no new information about the physical time beyond the resolution of the oscillator.

In light of the considerations above, the IEEE P1212 CSR draft standard[20] specifies the following format. Absolute time is represented by a 64-bit, fixed-point unsigned number. The binary radix point divides the 64-bit number into two 32-bit portions. The 32-bit most-significant portion represents seconds. The 32-bit least-significant portion represents a binary fraction of 1 second. The least-significant bit represents a period of approximately 233 picoseconds. The 64-bit unsigned value overflows approximately once every 136 years.

3.3 TIME SAMPLE FORMAT

Realtime clocks respond to the reception of a strobe broadcast by immediately, and simultaneously, sampling their own values and placing the respective results in their sample registers. This suggests that the sample register contents should have a format similar to that of the realtime clock value itself. The synchronization algorithm operates on the *relative differences* between sample values rather than the sample values themselves. Often hardware cost can be reduced by implementing only those sample register bits that have an impact on the worst case computations of the differences between sample values. The worst case depends on a number of factors.

First, the sample register must contain sufficient most-significant bits to support the computation of the worst case synchronization error. The synchronization error is the difference between the sample time value and the reference time value. When the source of the sample time value drifts in one direction and the source of the reference time value drifts in the other, the impact on the synchronization error between them is additive. Thus, sufficient bits are needed to support representation of the sum of the magnitudes of the worst case errors from each. And since the synchronization error may be either positive or negative, an additional bit is needed to support the sign of the difference.

Second, the sample register must contain sufficient most-significant bits to support the worst case initialization error. Since the realtime clock operates asynchronously to the software initializing it, it is often difficult to initialize the realtime clock value at a precise moment with respect to the value of other realtime clocks. Initialization sets the realtime clock to a value in the neighborhood of the reference time. The synchronization algorithm then makes the final adjustment as if the error was due to drift.

Third, the sample register may need to contain sufficient most-significant bits to support a faulty realtime clock synchronization error threshold A faulty realtime clock can tick erratically, or more commonly, simply fails to tick at all. If the number of missing or extra ticks is small, the fault may look like realtime clock drift. As the number of missing or extra ticks increases, a point is reached where the synchronization error cannot be considered drift, given the clock oscillator specifications. The synchronization error is compared against this threshold to detect faulty realtime clocks.

The sample register is designed to provide very accurate measurement of synchronization error. It provides more resolution in its least-significant bits than is usually necessary for detecting a faulty realtime

[20]See IEEE P1212 Draft 4.0, September 22, 1990, sec. 4.6.17, pp. 64-66 and sec. A-7, p. A41. This is an unapproved draft. Do not specify or claim conformance to this document.

clock. The threshold could be applied instead to the difference in the values obtained by reading the realtime clock values directly. Using the sample register usually improves synchronization algorithm software execution time since the sample register contents are obtained for drift compensation anyway.

The IEEE P1212 CSR draft standard specifies that sample registers have the same format as the most-significant 64-bit portion of the clock value registers.[21] It further specifies that unnecessary least-significant bits may be omitted from "adder and internal clock registers."[22] The intent was to avoid unnecessary complexity in the adder circuitry. All most-significant bits are required to encourage interoperability and improve fault detection execution speed.

3.4 SAMPLE PERIOD FORMAT

The selection of the period between strobe broadcasts, called the sample period, depends on the realtime clock synchronization resolution desired and on the rate at which the realtime clock values drift apart. The rate at which realtime clock values drift apart depends on the ability of the realtime clocks to track and compensate for differences in their oscillator frequencies.

Crystal oscillator frequency is a function of temperature. The worst case rate of frequency change occurs when power is first applied because applying power immediately raises the temperature of the crystal. A momentary interruption of power, such as when power is switched from one source to another, has little effect because there is insufficient time for significant temperature change of the crystal to occur.

Figure 21 shows crystal oscillator frequency change over time immediately after applying power to a military computer, an AN/UYK-44(V) "Military Reconfigurable Computer" (MRC).[23] Figure 22 shows similar results for a commercial computer, an NCR PC8, a clone of the IBM AT personal computer. Both computers show rapid frequency change for the first few seconds of operation, followed by gradual frequency change thereafter. The rapid frequency change is caused primarily by the rise in the temperature of the crystal itself. The gradual frequency change is caused by the rise in the temperature of the surrounding environment.

The maximum sample period that still guarantees that the realtime clock synchronization error remains within a given limit is computed from the *rate* of frequency change using the following equation

$$\text{sample period} = \frac{\text{clock error}}{\text{rate of freq change}} .$$

---

[21] See IEEE P1212 Draft 4.0, September 22, 1990, sec. 4.6.19, pp. 68-69. This is an unapproved draft. Do not specify or claim conformance to this document.

[22] See IEEE P1212 Draft 4.0, September 22, 1990, sec. A-7.3, p. A44. This is an unapproved draft. Do not specify or claim conformance to this document.

[23] Actually two AN/UYK-44(V) MRCs were tested, SN AA003 and SN AA033. The results were almost identical. The test data were recorded using a Hewlett Packard HP-5371A frequency modulation analyzer. It contains an internal temperature-controlled oscillator, which was allowed to operate for 3 days before conducting the experiments to assure stability.

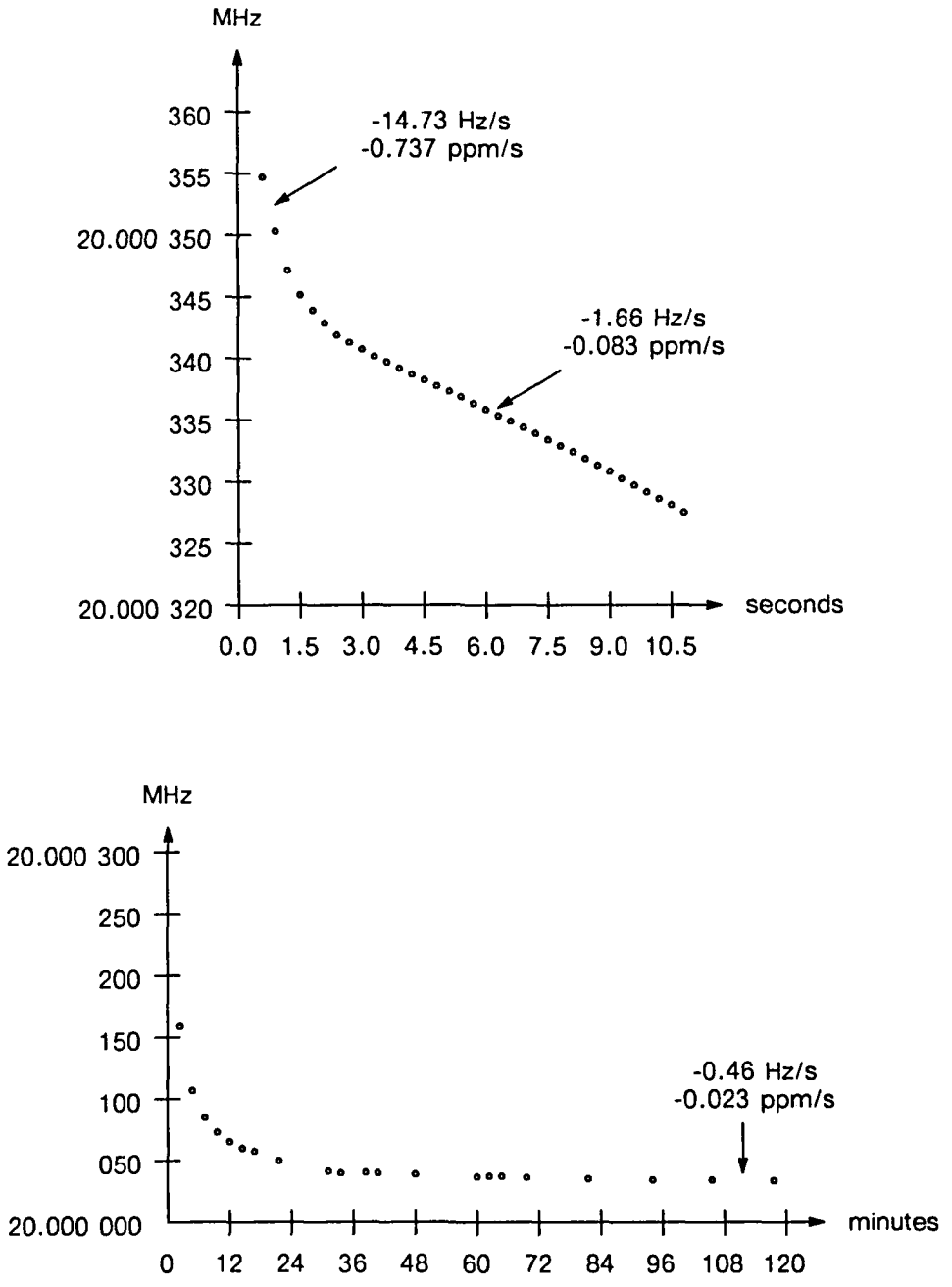
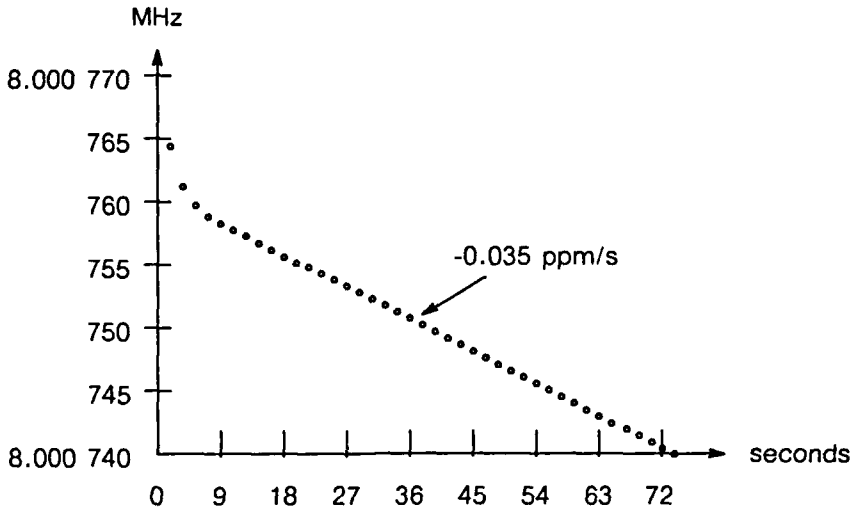


Figure 21. AN/UYK-44(V) oscillator frequency after power is applied.



NCR PC8 3279-0202 SN 17-16822438

Figure 22. NCR PC8 oscillator frequency after power is applied.

For example, if the maximum rate of frequency change is one part per million per second (ppm/s), a sample period of 1 second limits the maximum realtime clock synchronization error to 1 microsecond. Since the absolute value of the rate of frequency change generally decreases from the time power is applied, the sample period can increase correspondingly. The sample period may extend to several seconds or even to a minute.

One could represent a sample period in the same format as used for the realtime clock value. Such an approach, however, would impose unnecessary complexity on the hardware implementation of a sample period timer for two reasons. First, the range of reasonable sample periods makes it unnecessary to implement many of the most-significant and least-significant bits present in the realtime clock value format. There is no need to support sample periods greater than a minute. There is also no need to support resolution in the sample period finer than usable by the fault-tolerant mechanism that selects a single source of strobes. Second, using the realtime clock value format to represent the sample period requires that the sample period timer hardware decrement by subtracting the tick period on each cycle of the oscillator. The logic is greatly simplified by designing a timer that decrements by one on each cycle of the oscillator. This suggests an alternative format that represents the sample period in the units of local realtime clock oscillator ticks rather than the units of realtime clock value. Given a format where the least-significant bit represents one local realtime clock oscillator tick unit, the range of reasonable sample periods is easily represented within 32 bits.

The IEEE P1212 CSR standard[24] does not specify a sample period register because strobes can be scheduled by operating system software.

[24]See IEEE P1212 Draft 4.0, September 22, 1990. This is an unapproved draft. Do not specify or claim conformance to this document.

4.0 RATE ADJUSTMENT ALGORITHMS

The rate adjustment algorithm is the method used to adjust the rate of a realtime clock given its current and previously sampled synchronization error. The synchronization error of a realtime clock is the difference between the realtime clock value and the reference time value. The reference time is the measure of time assumed to be correct for the purposes of realtime clock synchronization. Methods of determining reference time were presented in section 1.4. Variable rate realtime clock hardware implementations were presented in sections 2.2 and 2.3. This section considers the algorithms used to control the variable rate realtime clock hardware. The discussion centers on the IEEE P1212 Control Status Register draft standard realtime clock model.[25] It is based on the phase accumulation method and is depicted in figure 20.

4.1 RATE ADJUSTMENT WITHOUT ALGORITHM EXECUTION DELAY

Synchronizing the realtime clock to the reference time is equivalent to obtaining and maintaining the minimum synchronization error between the two. It is useful to think of the problem in terms of synchronization error as a function of time. The value of the synchronization error function at a particular point in time represents the error in the realtime clock value at that time. The slope or derivative of the synchronization error function at a particular point in time represents the error in realtime clock rate at that time.

Rate adjustment algorithms are concerned with both the value and the rate of the realtime clock. If only the rate were corrected, whatever value error existed would continue to exist forever. If only the value were corrected, then, by definition, this would not be a rate adjustment algorithm with all the associated benefits described in section 1.2.

The rate adjustment algorithm designer has some freedom in selecting how quickly synchronization is obtained. Synchronization must be obtained quickly enough to keep pace with frequency variations in the realtime clock oscillator. Typical crystal oscillator performance was illustrated in figures 21 and 22. On the other hand, synchronization should not be obtained so rapidly that it adversely affects application computations dependent on realtime clock rate, such as time interval measurements.

Consider a realtime clock that is sampled and immediately adjusted at the expiration of each sample period. In other words, there is no delay between the time the strobe takes the sample and the time it is processed by the rate adjustment algorithm. The case where there is delay will be presented later. It is further assumed that all sample periods are of the same duration and that the realtime clock oscillator is stable. The problem is to design the rate adjustment algorithm such that synchronization is achieved exactly within one sample period.

The solution is shown graphically in figure 23. Points A and D plot the synchronization errors computed from the previous and current samples, respectively. If no adjustment were made in either the realtime clock value or rate, the synchronization error function would continue increasing at the same slope and the next sample would be expected at point H. The problem is to adjust the rate so it appears instead at point E. It is given that all sample periods have the same duration, so vector $AB = CE$. Choose

[25] See IEEE P1212 Draft 4.0, September 22, 1990, sec. A-7.3, p. A44. This is an unapproved draft. Do not specify or claim conformance to this document.

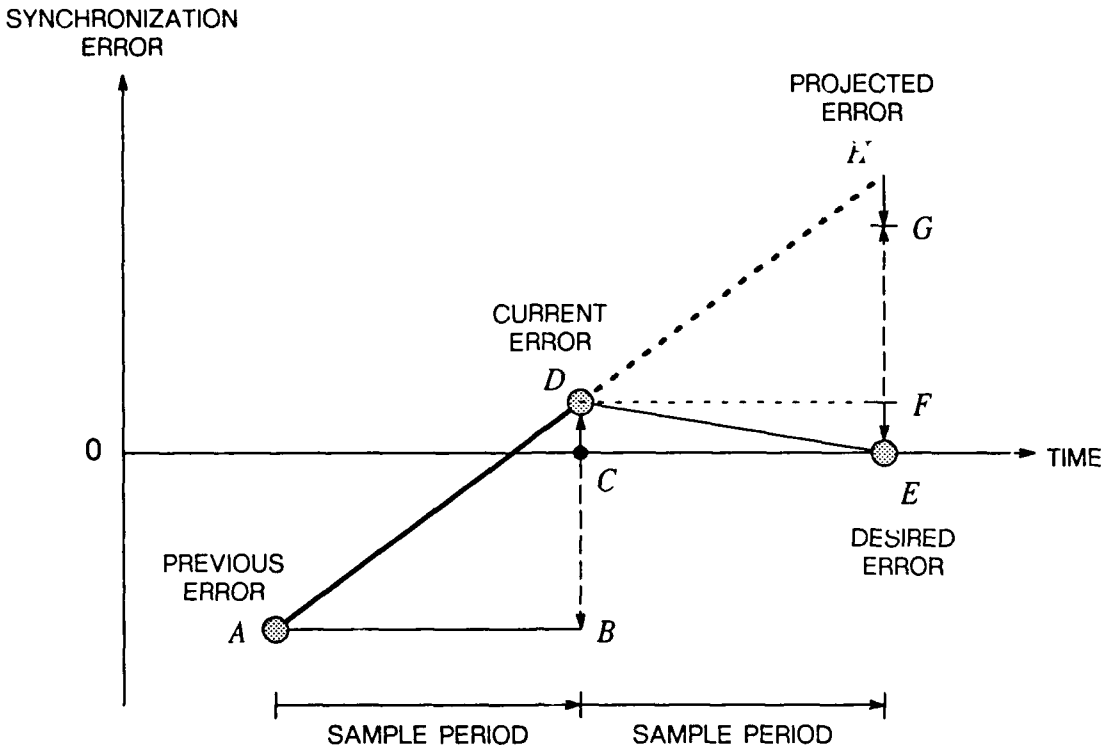


Figure 23. Geometric derivation of rate adjustment.

point G on FH such that $BC = FG$. By geometry it can be shown that $CD = EF = GH$. As previously stated, the rate adjustment algorithm is concerned with both the value and the rate of the realtime clock. The rate adjustment therefore consists of two components. The first rate component, $FE \div CE$, cancels the effects of the value offset CD over the period CE. The second rate component, $HF \div CE$, cancels the effects of the rising slope $DB \div AB$ over the period CE. The total rate adjustment is the sum of these two components, that is, $HE \div CE$. Note that since these components are computed from synchronization error, they represent the amount by which the realtime clock rate is *adjusted*, not the absolute realtime clock rate desired.

Knowing the amount of realtime clock rate adjustment, the next step is to apply it to the phase accumulation variable rate realtime clock hardware. Between samples, the rate at which error accumulates per unit time is assumed to be constant. For the samples at A and D, this rate is represented by the slope of the line AD. As shown by the similar triangles in figure 24, the tick error per tick period is equivalent to the sample error per sample period because they involve the same slope. Since all the respective triangles are similar, figure 23 can be used to derive the tick adjustment by rescaling it such that all references to the sample period are replaced with corresponding references to the tick period. This is shown in figure 25. Since the rate adjustment was a change in slope of HE per sample period CE, the tick adjustment, having the same change in slope, is $H'E'$ per tick period $C'E'$. To correct the phase accumulation variable rate realtime clock thus requires a change in the magnitude of the tick period by the amount $H'E'$. By vector addition, paying careful attention to the sense of the vectors:

$$H'E' = H'G' + G'F' + F'E' = 2 \times D'C' + C'B' = -(2 \times C'D' - C'B')$$

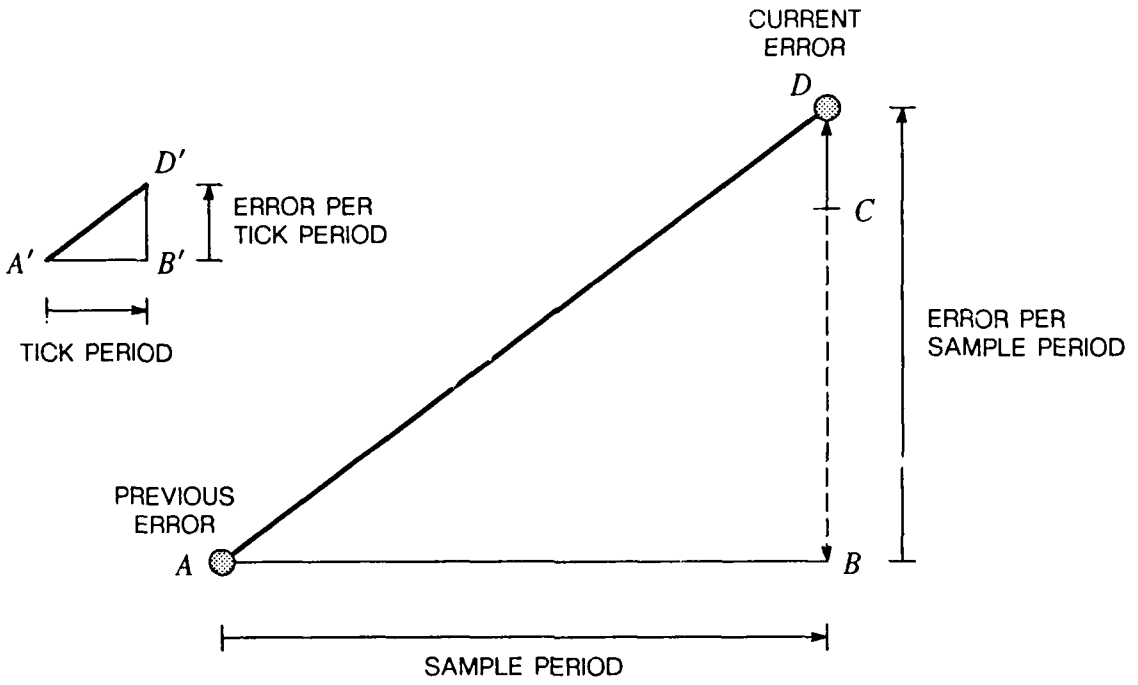


Figure 24. Geometric relationship between sample and tick errors.

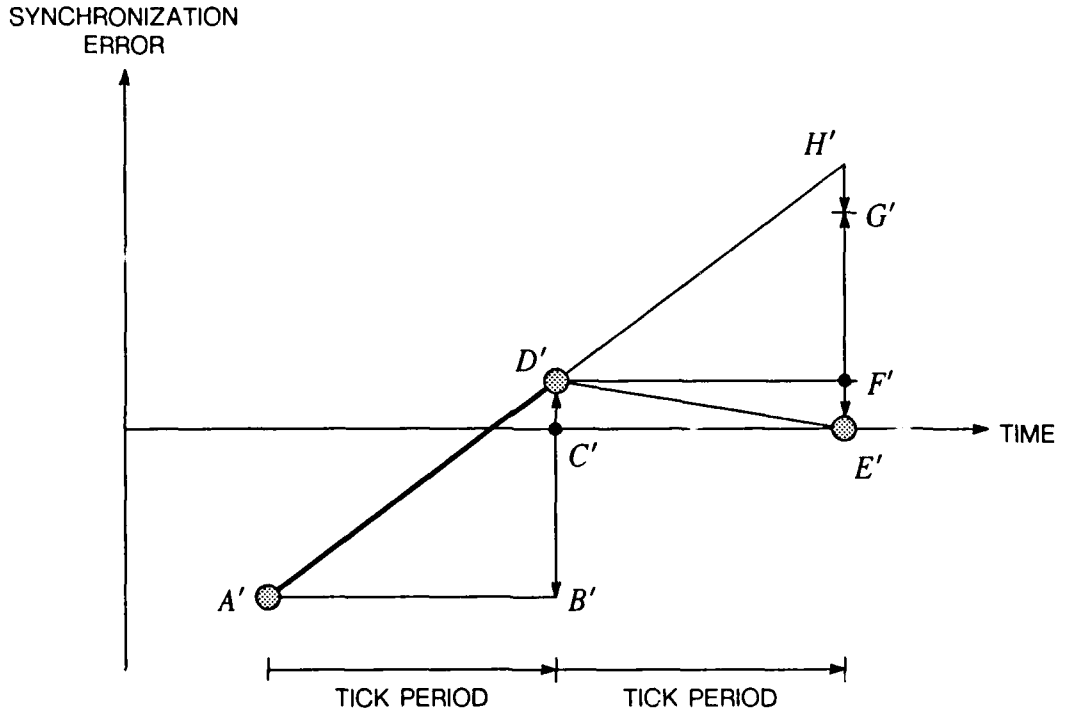


Figure 25. Geometric derivation of tick adjustment.

Since the tick period $A'B'$ has the same duration as the tick period $C'E'$, division of the equation above yields:

$$\frac{H'E'}{C'E'} = -\frac{2 \times C'D' - C'B'}{A'B'}$$

By the proportional relationships of similar triangles such as shown in figure 24:

$$\frac{H'E'}{C'E'} = -\frac{2 \times C'D' - C'B'}{A'B'} = -\frac{2 \times CD - CB}{AB}$$

This is the amount *per tick* that a tick must be *adjusted* to obtain convergence at point E. Thus, for a phase accumulation variable rate realtime clock with no rate adjustment delay and regular sample periods, the required proportional change in tick period is

$$\text{new tick period} = \text{old tick period} \times \left(1 - \frac{2 \times \text{current sample error} - \text{previous sample error}}{\text{sample period}}\right)$$

4.2 RATE ADJUSTMENT WITH ALGORITHM EXECUTION DELAY

It is critical that the local realtime clock be sampled immediately upon detection of a strobe. This is normally accomplished using hardware as described in section 1.3. It is not critical, however, that the adjustment algorithm immediately process the sample once it is obtained. This is one of the major advantages of the strobe technique.

Figure 26 shows the impact of a delay in rate adjustment algorithm execution. Point X projects the synchronization error expected when the rate adjustment algorithm completes execution. As the delay increases, point X approaches point H.

The equation derived above assumes that the rate adjustment algorithm is executed immediately after the strobe samples the realtime clock value and that it requires negligible execution time. It may seem that the delay problem can be solved simply by deriving a new equation to obtain synchronization at point E by adjusting the rate at point X rather than at point D. But as the delay increases, the slope of the line XE becomes increasingly steeper. There are two dangers. First, the rate adjustment, depicted by the slope, may become more rapid than acceptable to the application software. Second, successively recurring delays can cause the slope to oscillate up and down with increasingly severe swings even though the samples obtained by the strobes look good. Figure 27 illustrates this situation.

Assuming an equation that targets synchronization at point E, the magnitude of the synchronization error oscillation increases when the delay is greater than half the sample period CE. Equivalently, the magnitude increases when the sample period is less than twice the delay.[26] Although delays cause

[26]The two-to-one relationship suggests affinity to the Nyquist sampling principle. The Nyquist sampling principle specifies the minimum frequency at which one must sample to avoid losing signal information. Since the Nyquist rate is a constant for a given signal, the maximum period between samples is also constant. The problem of variations in execution delay, described above, is also one of limiting the maximum period between samples. In the latter case, the information contained in the signal is the frequency of the uncorrected clock synchronization error.

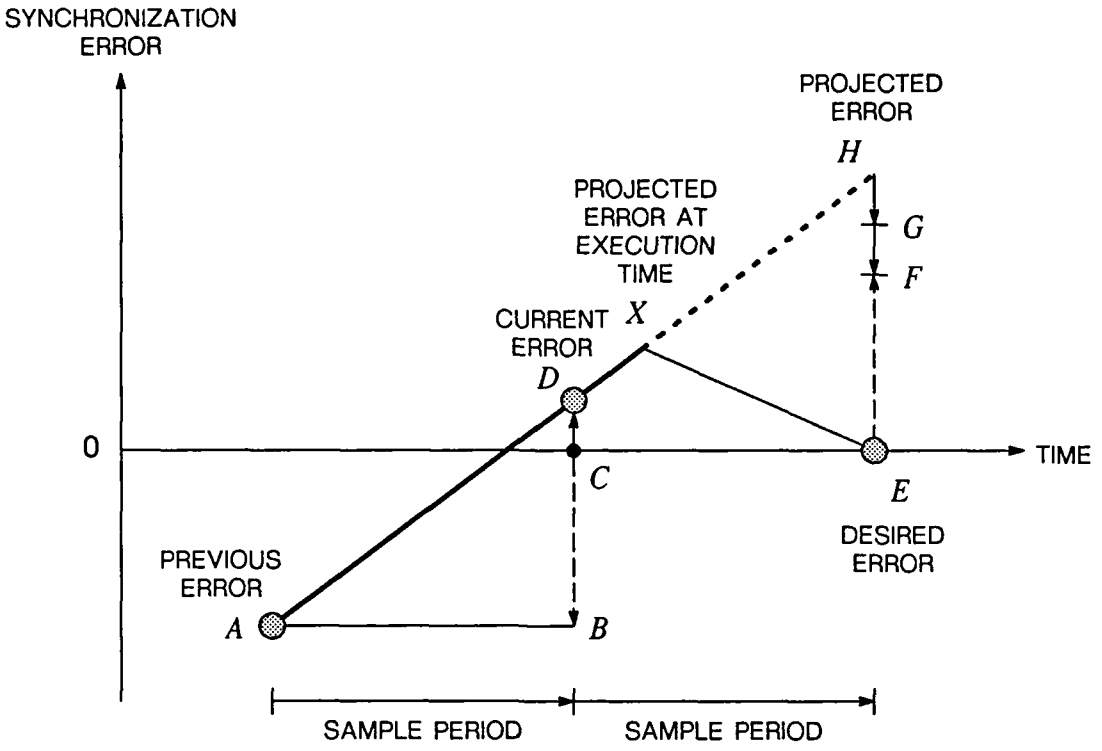


Figure 26. Geometric affect of rate adjustment execution delay.

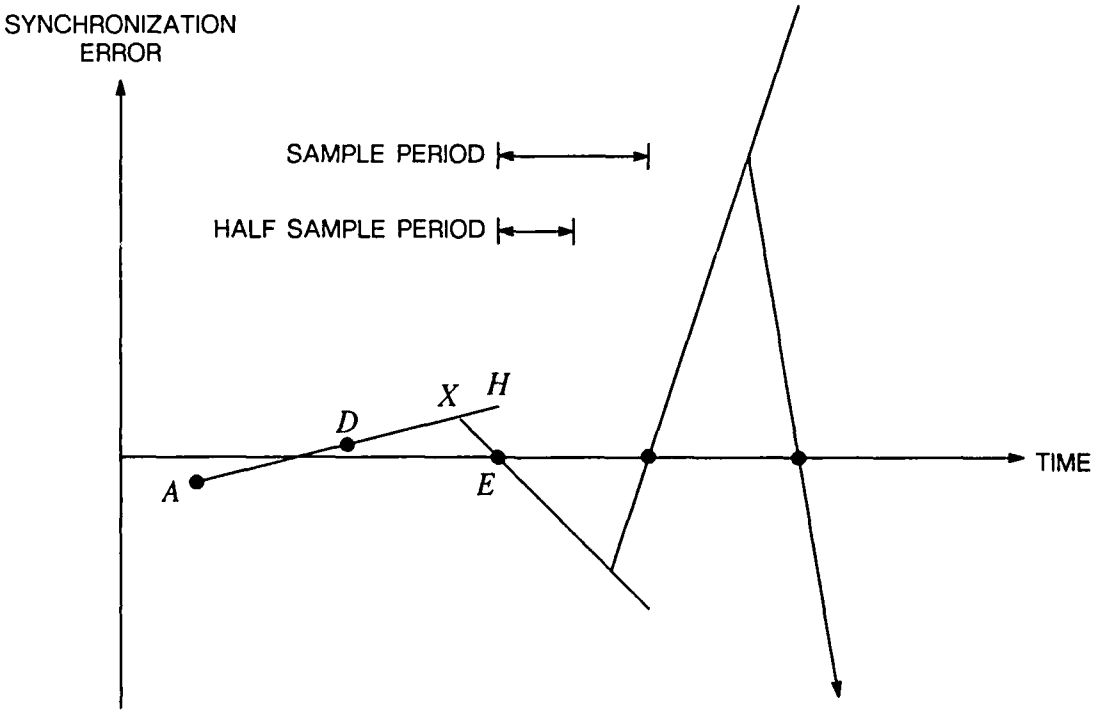


Figure 27. Geometric representation of unstable rate adjustment.

synchronization error oscillations, the designer can assure stability by selecting a sample period that is at least no smaller than twice the maximum expected delay. If this is not possible, the designer can target synchronization at a point S beyond point E, as shown by figure 28.

The equation needed to converge at an arbitrary time in the future, point S, can be derived with the help of figure 28. Points A and D plot the synchronization errors computed from the previous and current samples, respectively. The problem is to adjust the rate so the realtime clock is synchronized at point S. The sample period is AB. The point V is selected such that $UV = AB$. The rate adjustment is delayed by CU. The rate adjustment component $YW \div UV$ cancels the effects of the value offset UX over the period US. The rate component $ZY \div UV$ cancels the effects of the rising slope $DB \div AB$. The sum of the two compoents is thus

$$\frac{ZW}{UV} = \frac{ZY + YW}{UV} = \frac{DC + CB + YW}{AB} = -\frac{CD - CB + WY}{AB} .$$

Only WY is left to be determined. By the similar triangles ΔWYX and ΔXUS

$$\frac{WY}{XY} = \frac{UX}{US} .$$

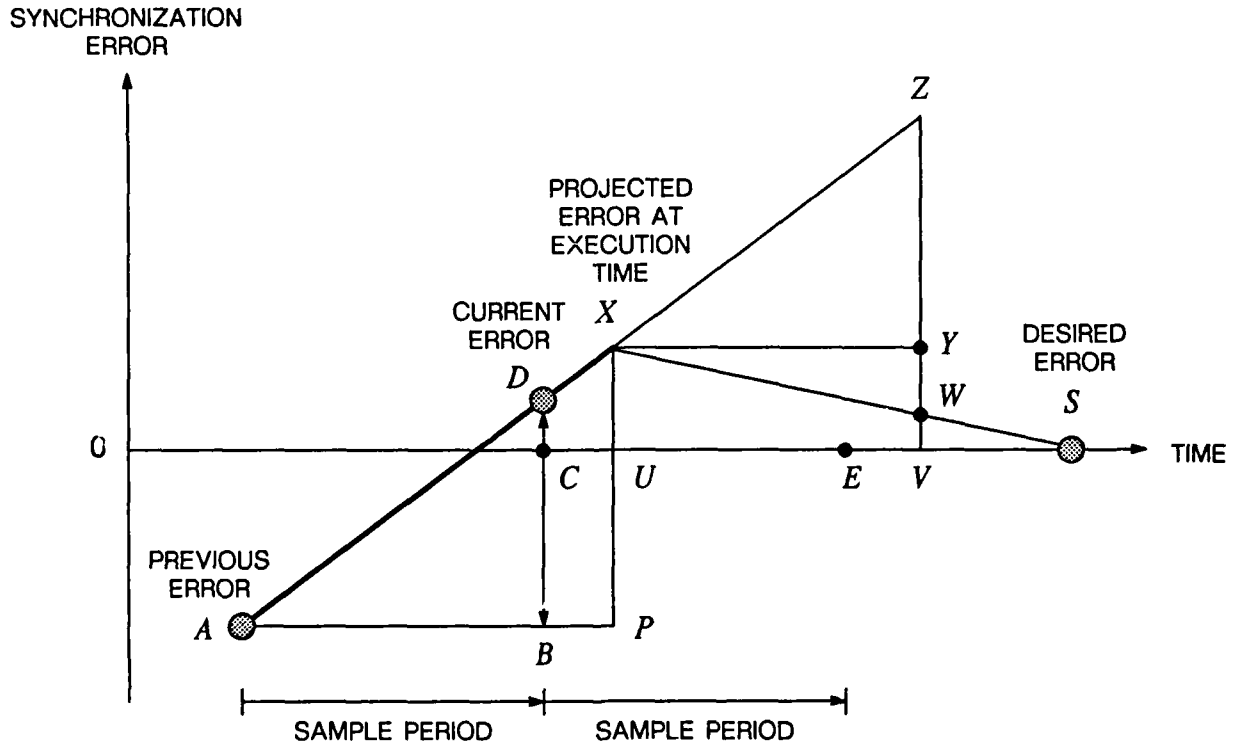


Figure 28. Geometric derivation of rate adjustment with delay.

Solving for WY

$$WY = \frac{UX}{US} \times XY = \frac{PX - PU}{US} \times AB = \frac{PX + CB}{US} \times AB .$$

By the similar triangles ΔXPA and ΔDBA

$$\frac{PX}{AP} = \frac{BD}{AB} .$$

Solving for PX

$$PX = \frac{BD}{AB} \times AP = \frac{BC + CD}{AB} \times AP = \frac{CD - CB}{AB} \times (AB + CU) .$$

Substituting PX into the equation for WY

$$WY = \frac{PX + CB}{US} \times AB = \frac{\frac{CD - CB}{AB} \times (AB + CU) + CB}{US} \times AB$$

$$= \frac{(CD - CB) + (CD - CB) \times \frac{CU}{AB} + CB}{US} \times AB = \frac{CD + (CD - CB) \times \frac{CU}{AB}}{US} \times AB$$

$$= \frac{CD \times AB + (CD - CB) \times CU}{US} .$$

Finally, solving for $ZW \div UV$

$$\frac{ZW}{UV} = - \frac{CD - CB + WY}{AB} = - \frac{CD - CB + \frac{CD \times AB + (CD - CB) \times CU}{US}}{AB}$$

$$= - \frac{CD - CB}{AB} - \frac{CD \times AB + (CD - CB) \times CU}{US \times AB}$$

where: CD = current sample synchronization error
CB = previous sample synchronization error
CU = time between current sample taken and rate adjustment
AB = time between previous and current samples taken
US = time between rate adjustment and synchronization.

The first term corrects the realtime clock rate error. It is independent of the execution delay CU. The second term corrects the realtime clock value error. It depends on CU because longer delay causes greater accumulation of synchronization error. The new tick period for the phase accumulation method is

$$\text{new tick period} = \text{old tick period} \times \left(1 - \frac{ZW}{UV}\right)$$

The more restrictive equation derived in section 4.1 is obtained from the equations above by setting $CU = 0$ and $US = AB$.

In practical systems, it is often impossible to predict the exact delay CU at design time. Higher priority activity and blocking may cause variability in CU. In such cases, CU is determined at execution time by subtracting the sample time from the current time.

To achieve synchronization at point S, point S must be treated as an absolute point in time. While approaching point S, the interval US decreases by the amount AB within the rate adjustment algorithm as each new sample is processed. If US were taken instead as a constant, synchronization would be obtained in the limit rather than at an absolute time. The convergence of synchronization in the limit is shown in figure 29.

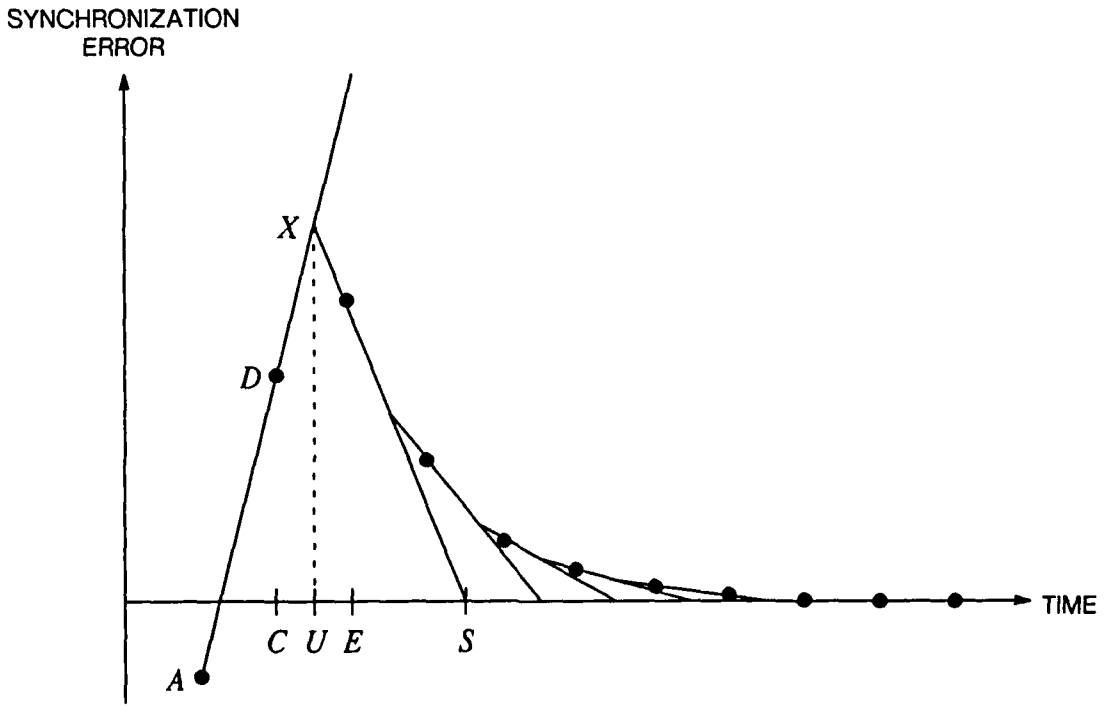


Figure 29. Geometric representation of synchronization in the limit.

BIBLIOGRAPHY

Arvind, K. 1990. "A New Probabilistic Algorithm for Clock Synchronization," Dept. of Computer and Information Science, Univ. of Massachusetts, Amherst, MA 01003.

Cole, R., and C. Forcroft. 1988. "An Experiment in Clock Synchronization," *Computer Journal*, vol. 31, no. 6, pp. 496-502.

Dolev, D., J. Y. Halpern, and R. H. Strong. 1986. "On the Possibility and Impossibility of Achieving Clock Synchronization," *Journal of Computer and System Science*, vol. 32, no. 2, pp. 230-250.

Gusella, R., and S. Zatti. 1989. "The Accuracy of the Clock Synchronization Achieved by TEMPO in Berkeley UNIX 4.3BSD," *IEEE Trans. Software*, vol. 15, no. 7, pp. 847-853.

Halpern, J. Y., B. Simons, R. Strong, and D. Dolev. 1984. "Fault-Tolerant Clock Synchronization," *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, ACM, Vancouver, B.C., Canada, pp. 89-102.

Harrington, E. A. 1978. "Synchronization Techniques for Various Switching Network Topologies," *IEEE Trans. Communications*, vol. COM-26, no. 6, pp. 925-932.

Inter-Range Instrumentation Group. 1987. *IRIG Standard 205-87: Parallel Binary and Parallel Binary Coded Decimal Time Code Formats*; Secretariat, Range Commanders Council, White Sands Missile Range, New Mexico 88002.

Jefferson, D. R. 1985. "Virtual Time," *ACM Trans. Prog. Languages*, vol. 7, no. 3, pp. 404-425.

Kopetz, H., and W. Ochsenreiter. 1987. "Clock Synchronization in Distributed Real-Time Systems," *IEEE Trans. Computers*, vol. C-36, no. 8, pp. 933-940.

Kopetz, H., A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger. 1989. "Distributed Fault-Tolerant Real-Time Systems: The Mars Approach," *IEEE Micro*, vol. 9, no. 1, pp. 25-40 (clock sync. pp. 31-32).

Krishna, C. M., and I. S. Bhandari. 1988. "On the Graceful Degradation of Phase-Locked Clocks," *Proceedings Real-Time Systems Symposium*, IEEE, pp. 202-211.

Lamport, L. 1978. "Time, Clocks, and the Ordering of Events in a Distributed System," *Comm. ACM*, vol. 21, no. 7, pp. 558-565.

Lamport, L. 1984. "Using Time Instead of Timeout for Fault-Tolerant Distributed Systems," *ACM Trans. Prog. Languages*, vol. 6, no. 2, pp. 254-280.

Lamport, L., and P. M. Melliar-Smith. 1984. "Byzantine Clock Synchronization," *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, ACM, Vancouver, B.C., Canada, Aug. 27-29, pp. 68-74.

Lamport, L., and P.M. Melliar-Smith. 1985. "Synchronizing Clocks in the Presence of Faults," *Journal ACM*, vol. 32, no. 1, pp. 52-78.

Lundelius, J., and N. Lynch. 1984. "A New Fault-Tolerant Algorithm for Clock Synchronization," *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, ACM, Vancouver, B.C., Canada, Aug. 27-29, pp. 75-88.

Mills, D. L. 1990. *Network Time Protocol (Version 3) Specification, Implementation, and Analysis*; Dept. Electrical Engr., Univ. Delaware, Newark, Delaware 19716.

Molle, M. L., and L. Kleinrock. 1985. "Virtual Time CSMA: Why Two Clocks Are Better than One," *IEEE Trans. Communications*, vol. COM-33, no. 9, pp. 919-933.

National Semiconductor. 1989. "Calibration of the DP8570A Family," *Advanced Peripherals: Real Time Clock Handbook*; National Semiconductor, 2900 Semiconductor Dr., Santa Clara, CA 95052-8090, application note AN-588, pp. 2.36-2.40.

Ramanathan, P., K. G. Shin, and R. W. Butler. 1990. "Fault-Tolerant Clock Synchronization in Distributed Systems," *IEEE Computer*, vol. 23, no. 10, pp. 33-42.

Robinson, D. C. 1989. "Using Bus Level Time Code Processors to Synchronize Multiple Signal Processing Stations," *International Telemetering Conference*, San Diego, vol. 25, pp. 337-344; also available from Bancomm (Datum, Inc.), 6541 Via del Oro, San Jose, CA 95119.

Saltzberg, B. R., and H. M. Zydney. 1975. "Network Synchronization," *Bell Sys. Tech. Journal*, vol. 54, no. 5, pp. 879-892.

Srikanth, T. K., and S. Toueg. 1987. "Optimal Clock Synchronization," *Journal ACM*, vol. 34, no. 3, pp. 626-645.

Volz, R. A., and T. N. Mudge. 1987. "Instruction Level Timing Mechanisms for Accurate Real-Time Task Scheduling," *IEEE Trans. Computers*, vol. C-36, no. 8, pp. 988-993.

Volz, R. A., and T. N. Mudge. 1987. "Timing Issues in the Distributed Execution of Ada Programs," *IEEE Trans. Computers*, vol. C-36, no. 4, pp. 449-459.

Volz, R. A., L. R. Sha, and D. R. Wilcox. "Maintaining Global TIme in Futuregus+," accepted for publication in *Real TIme Systems Journal*.

Wilcox, D. R. 1989. "Periodic Phase Adjustment Distributed Clock Synchronization in the Hard Realtime Environment." NOSC TR 1310, Naval Ocean Systems Center, San Diego, CA.

APPENDIX A—CRYSTAL OSCILLATOR TEST DATA

To obtain a feel for the nature and relative magnitude of the drift of crystal oscillators within densely packed electronic equipment designed for the severe military environment, the performance of the crystals within AN/UYK-44(V) Modular Reconfigurable Computers (MRCs) were tested. The test data were recorded using a Hewlett Packard HP-5371A frequency modulation analyzer. It contains a temperature-controlled internal oscillator, which was allowed to run for 3 days before conducting the experiments to assure its stability.

The following table shows the drift of two AN/UYK-44(V) MRC oscillators for the first 18 seconds of operation immediately after applying power after remaining off overnight. No significance should be attributed to the values missing from the table.

Time Seconds	SN AA033 No program Frequency MHz		SN AA003 No program Frequency MHz	
0.3	19.732	327 87	20.627	020 28
0.6	19.999	681 36	20.000	354 71
0.9		679 25		350 29
1.2		678 17		347 17
1.5		677 09		345 15
1.8		676 11		343 88
2.1		675 53		342 83
2.4		674 99		341 88
2.7		674 45		341 31
3.0		673 93		340 75
3.3		673 41		340 17
3.6		672 92		339 68
3.9		672 39		339 19
4.2		671 85		338 71
4.5		671 28		338 24
4.8		670 75		337 77
5.1		670 17		337 29
5.4		669 61		336 83
5.7		669 01		336 33
6.0		668 43		335 85
6.3		667 83		335 37
6.6		667 21		334 89
6.9		666 60		334 41
7.2		666 00		333 89
7.5				333 40
7.8				332 89
8.1				332 39
8.4				331 85
8.7		662 84		331 32
9.0		662 28		330 79

Time Seconds	SN AA033 No program Frequency MHz	SN AA003 No program Frequency MHz
9.3	661 56	330 27
9.6	660 95	329 71
9.9	660 31	329 16
10.2	659 68	328 61
10.5	659 04	328 09
10.8	658 40	327 51
11.1	657 79	
11.4	657 15	
11.7	656 55	
12.0	655 93	325 31
12.3	655 28	324 77
12.6	654 69	324 19
12.9	654 00	323 65
13.2	653 48	323 08
13.5	652 87	322 52
13.8	652 29	321 95
14.1	651 69	321 39
14.4	651 11	320 83
14.7	650 49	320 25
15.0	649 95	319 69
15.3	649 32	319 12
15.6	648 77	
15.9	648 16	
16.2	647 63	
16.5	647 05	316 83
16.8	646 47	316 28
17.1	645 93	315 71
17.4	645 37	315 12
17.7	644 83	314 57
18.0	644 28	314 03

The following table shows the drift of an AN/UYK-44(V) MRC oscillator for the first 2 hours of operation immediately after applying power after remaining off overnight. No significance should be attributed to the values missing from the table.

Time Seconds	SN AA033 With program Frequency MHz	
7.2	20 000	299 311 1
14.4		299 285 0
21.6		287 531 1
28.8		274 938 3
36.0		262 991 7
43.2		251 998 3
50.4		241 998 9
57.6		232 821 7
64.8		224 321 7
72.0		216 424 4
79.2		209 127 8
86.4		202 314 4
93.6		195 924 4
100.8		189 731 1
108.0		184 002 8
115.2		178 695 0
122.4		173 719 4
129.6		168 622 8
136.8		163 630 6
144.0		159 188 3
151.2		154 895 6
158.4		151 051 1
165.6		147 396 1
172.8		144 015 6
180.0		140 788 9
187.2		137 613 9
194.4		134 651 7
201.6		131 974 4
208.8		129 346 1
216.0		126 986 7
223.2		124 501 1
230.4		122 221 1
237.6		119 995 6
244.8		117 005 6
252.0		115 837 8
259.2		113 883 9
266.4		112 866 1
273.6		110 403 3
280.8		108 699 4
288.0		107 029 4
295.2		105 402 2
302.4		103 985 6
309.6		102 587 8
316.8		101 225 6

Time Seconds	SN AA033 With program Frequency MHz
324.0	099 912 8
331.2	098 678 0
338.4	097 455 0
345.6	096 519 4
352.8	095 679 4
360.0	094 408 9
367.2	093 076 1
374.4	091 943 9
381.6	090 884 4
388.8	089 905 6
396.0	089 000 0
403.2	088 205 6
410.4	087 395 6
417.6	086 562 8
424.8	085 827 8
432.0	084 990 6
439.2	084 152 2
446.4	083 501 1
453.6	082 828 9
460.8	082 287 2
468.0	081 572 2
475.2	080 730 6
482.4	079 993 9
489.6	079 307 2
496.8	078 606 7
504.0	077 982 2
511.2	077 562 2
518.4	077 148 9
525.6	076 548 9
532.8	075 811 7
540.0	075 166 7
547.2	074 564 4
554.4	074 247 8
561.6	073 836 7
568.8	073 313 3
576.0	072 965 0
583.2	072 470 6
590.4	071 872 2
597.6	071 461 7
604.8	071 068 3
612.0	070 740 0
619.2	070 328 9
626.4	069 894 4
633.6	069 493 3
640.8	069 096 1

Time Seconds	SN AA033 With program Frequency MHz
648.0	068 702 2
655.2	068 560 0
662.4	068 284 4
669.6	067 905 0
676.8	067 666 1
684.0	067 181 7
691.2	066 739 4
698.4	066 383 9
705.6	066 026 1
712.8	065 886 7
720.0	065 716 1
792.0	063 103 3
864.0	060 013 3
1008.0	057 455 0
1224.0	052 893 9
1296.0	050 006 7
1584.0	046 335 0
2016.0	041 355 0
2023.2	041 669 4
2030.4	041 723 9
2037.6	041 536 7
2044.8	041 271 1
2052.0	040 857 2
2059.2	040 880 6
2066.4	040 790 0
2073.6	040 716 1
2080.8	040 472 8
2088.0	040 297 8
2095.2	040 267 8
2102.4	040 270 0
2109.6	040 144 4
2116.8	040 280 6
2124.0	039 897 8
2131.2	039 574 4
2138.4	039 485 0
2145.6	039 407 2
2152.8	039 452 8
2160.0	039 769 4
2167.2	040 102 8
2174.4	040 473 9
2181.6	040 642 2
2188.8	041 254 4

Time Seconds	SN AA033 With program Frequency MHz
2196.0	041 598 9
2203.2	041 726 1
2210.4	041 832 2
2217.6	042 034 4
2224.8	042 205 0
2232.0	042 347 8
2304.0	042 245 0
2311.2	042 088 9
2318.4	042 041 7
2325.6	041 927 8
2376.0	040 981 1
2448.0	040 387 8
2520.0	040 084 4
2556.0	039 931 7
2563.2	039 928 3
2570.4	039 950 6
2577.6	039 884 4
2584.8	039 903 9
2592.0	039 870 6
2599.2	040 005 6
2606.4	039 975 6
2613.6	039 927 2
2952.0	039 196 7
3024.0	039 035 6
3600.0	036 988 9
3672.0	037 016 7
3744.0	037 560 0
3888.0	037 442 2
3960.0	037 153 9
40[illegible]0	036 480 6
4104.0	035 738 9
4248.0	035 030 6
4680.0	036 045 0
4752.0	035 617 2
5400.0	034 071 7
5472.0	034 534 4
6120.0	034 535 3

Time Seconds	SN AA033 With program Frequency MHz
6192.0	034 576 1
6840.0	033 301 1
6912.0	033 864 4
6984.0	034 911 7
7056.0	033 807 8
7128.0	032 217 2
7178.4	032 462 2

The following table shows the drift of an NCR PC8 Personal Computer oscillator for the first 54 seconds of operation immediately after applying power after remaining off overnight.

Time Seconds	Frequency MHz
1.8	8.000 764 413
3.6	761 212
5.4	759 679
7.2	758 787
9.0	758 250
10.8	757 725
12.6	757 225
14.4	756 667
16.2	756 133
18.0	755 612
19.8	755 091
21.6	754 795
23.4	754 292
25.2	753 800
27.0	753 289
28.8	752 776
30.6	752 273
32.4	751 757
34.2	751 238
36.0	750 727
37.8	750 186
39.6	749 672
41.4	749 148
43.2	748 628
45.0	748 116
46.8	747 591
48.6	747 070
50.4	746 564
52.2	746 052
54.0	745 529

INDEX

L

M

N

O

P

R

S

T

U

V

W

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1990	Final

4. TITLE AND SUBTITLE

BACKPLANE BUS DISTRIBUTED REALTIME CLOCK SYNCHRONIZATION

5. FUNDING NUMBERS

0602234N
RS34C76
ECB1
DN306243

6. AUTHOR(S)

D. R. Wilcox

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Ocean Systems Center
San Diego, CA 92152-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

NOSC TR 1400

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Naval Ocean Systems Center
San Diego, CA 92152-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

Code 41

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Authorized for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report presents methods of synchronizing the realtime clocks associated with a set of respective digital processing modules interconnected through a backplane bus. All the methods are based on the broadcast of a strobe signal over the backplane bus, which simultaneously samples all the realtime clocks being synchronized. The report describes and compares various methods of implementing realtime clocks, designed to permit their rate of advance with respect to physical time to be adjusted. It examines and makes recommendations on various interface standardization issues related to variable rate realtime clocks. It also gives and evaluates software algorithms for synchronizing variable rate realtime clocks.

14. SUBJECT TERMS

backplane interconnect
computer architecture

15. NUMBER OF PAGES

61

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAME AS REPORT

INITIAL DISTRIBUTION

Code 0012	Patent Counsel	(1)
Code 0144	R. November	(1)
Code 40	R. C. Kolb	(1)
Code 401	R. A. Wasilausky	(1)
Code 41	M. L. Crowley	(1)
Code 41	A. J. Justice	(1)
Code 412	R. M. Bergman	(1)
Code 412	P. A. Casey	(10)
Code 412	L. J. Core	(1)
Code 412	P. N. Pham	(1)
Code 412	L. A. Rasmus	(1)
Code 412	D. F. Romig	(1)
Code 412	D. G. Sheriff	(1)
Code 412	V. D. Tran	(1)
Code 412	D. R. Wilcox	(30)
Code 413	L. C. Anderson	(1)
Code 413	D. W. Brouhard	(1)
Code 413	R. E. Johnston	(1)
Code 552	H. D. Copeland	(1)
Code 6201	R. Eyres	(1)
Code 623	B. Gosse	(1)
Code 821	R. Duffy	(1)
Code 821	E. A. Howard	(1)
Code 952B	J. Puleo	(1)
Code 961	Archive/Stock	(6)
Code 964B	Library	(3)

Defense Technical Information Center
Alexandria, VA 22304-6145 (2)

Center for Naval Analyses
Alexandria, VA 22302-0268 (1)

Naval Air Development Center
Warminster, PA 18974-5000 (3)

Naval Surface Warfare Center
Silver Spring, MD 20903-5000 (1)

Fleet Combat Direction Systems
Support Activity
San Diego, CA 92147-5081 (1)

Naval Weapons Center
China Lake, CA 93555 (1)

Naval Weapons Support Center
Crane, IN 47522-5060 (1)

Synetics Corporation
El Cajon, CA 92019 (1)

NOSC Liaison Office
Washington, DC 20363-5100 (1)

Office of Naval Technology
Arlington, VA 22217-5000 (1)

Space & Naval Warfare Systems Command
Washington, DC 20363-5100 (4)

Naval Surface Warfare Center
Dahlgren, VA 22448-5000 (3)

Naval Sea Systems Command
Washington, DC 20362-5101 (1)

Naval Underwater Systems Center
Newport, RI 02841-5047 (1)

National Semiconductor Corporation
Santa Clara, CA 95052-8090 (1)

Apple Computer
Cupertino, CA 95014 (1)

Carnegie Mellon University
Software Engineering Institute
Pittsburgh, PA 15213 (2)

Carnegie Mellon University
Dept of Computer Science
Pittsburgh, PA 15212 (1)

University of Virginia
Dept of Computer Science
Charlottesville, VA 22903 (1)

University of Delaware
Dept of Electrical Engineering
Newark, DE 19716 (1)

Texas A&M University
Dept of Computer Science
College Station, TX 77843-3112 (1)

Unisys Corporation
St Paul, MN 55164-0525 (1)